

Softwaretechnik

Prof. Dr. Rainer Koschke

Fachbereich Mathematik und Informatik
Arbeitsgruppe Softwaretechnik
Universität Bremen

Wintersemester 2012/13

Überblick I

Vorbemerkungen

Entwicklungsprozesse

Software-Metriken

Empirische Softwaretechnik

Kontrollierte Experimente

Statistik bei kontrollierten Experimenten

Kosten- und Aufwandsschätzung

Entwurfsmuster

Überblick II

Komponentenbasierte Entwicklung

OSGI als Beispiel eines Komponentenmodells

Software-Produktlinien

Modellgetriebene Softwareentwicklung

Vorbemerkungen

Vorbemerkungen

Themen der Vorlesung

Übersicht

Termine

Übungen und Ressourcen

Scheinbedingungen

Lehrbücher

- Entwicklungsprozesse
- Metriken
- Kosten- und Aufwandsschätzung
- Komponentenbasierte Entwicklung
- Entwurfsmuster (Schwerpunkt Parallelität)
- Software-Architektur
- Modellgetriebene Softwareentwicklung
- Software-Produktlinien
- Fortgeschrittenes Testen
- Empirie in der Softwaretechnik

Übungen finden ca. alle zwei Wochen alternierend zur Vorlesung am Mittwoch statt.

Zu einzelnen Themen werden möglicherweise externe Dozenten eingeladen. Das letzte Mal war beispielsweise der Chefarchitektur von OpenOffice von Sun Microsystems dabei.

Eingebettet in die Vorlesung gibt es zur Vertiefung des Softwareprojektmanagements einen praktischen Teil mit SESAM. SESAM ist ein Softwareprojektabenteuerspiel, bei dem Ihr in die Rolle eines Projektleiters schlüpft. Ihr bekommt ein Budget und harte Anforderungen an Qualität und Projektdauer. Dann könnt ihr Entwickler einstellen und beauftragen, verschiedene Aktivitäten zu übernehmen, um die Anforderungen zu erheben, die Architektur zu entwerfen, zu implementieren und zu testen. Es gewinnt, wer in den vorgegeben Grenzen der Dauer und Qualität bleibt. Jeder lernt dazu durch das selbstständige Spielen am SESAM-Simulator und durch Feedback von mir zu seinen Entscheidungen im Projekt. Die Aufgabe ist eine echte Herausforderung, bei der man viel lernen kann zur Durchführung eines Software-Projekts, ohne dass man Gefahr läuft, viel echtes Geld zu versenken.

Entwicklungsprozesse

- alternative Software-Entwicklungsprozesse (z.B. Clean-Room und Extreme Programming)
- Capability Maturity Model, Spice und Bootstrap
- Prozessverbesserungen
- Persönlicher Prozess
- Literatur: Balzert (2008); Bunse und von Knethen (2002); Kneuper (2006); Siviý u. a. (2007)

Softwaremetriken

- was ist eine Metrik?
- Messtheorie
- Skalen
- Prozess-, Produkt- und Ressourcenmetriken
- Literatur: Fenton und Pfleeger (1998)

Kosten- und Aufwandsschätzung

- insbesondere Function-Points und CoCoMo I und II
- Literatur: Poensgen und Bock (2005); Boehm u. a. (2000)

Komponentenbasierte Entwicklung

- Eigenschaften, Vor- und Nachteile
- Komponentenmodell
- Schnittstellen und Kontrakte
- Managementfragen
- Rahmenwerke
- existierende Komponentensysteme, z.B. .NET, Microsoft DCOM, OLE, ActiveX, Sun Java und JavaBeans
- Literatur: Szyperski u. a. (2002)

Modellgetriebene Softwareentwicklung

- Ideen, Eigenschaften, Vor- und Nachteile
- Werkzeugunterstützung am Beispiel von Eclipse Open Architecture Ware
- Literatur: Stahl u. a. (2007)

Software-Architektur

- Entwurfsmuster
- Qualitätseigenschaften
- Analyse von Architekturen (insbesondere SAAM und ATAM)
- Literatur: Buschmann u. a. (1996); Gamma u. a. (2003); Bass u. a. (2003); Hofmeister u. a. (2000)

Software-Produktlinien

- Definition und Beispiele
- Vor- und Nachteile
- Practice Areas
- Einführung von Produktlinien
- Ansätze zur technischen Realisierung
- Beschreibungen und Notationen (z.B. Feature-Graphen)
- Besonderheiten beim Requirementsengineering, Konfigurationsmanagement und Test
- Konfiguration von Produktlinien
- Literatur: Clements und Northrop (2001)

Empirisches Software-Engineering

- Empirische Forschung in der Softwaretechnik
- Methoden
- Literatur: Endres und Rombach (2003); Prechelt (2001); Yin (2003)

Allgemeine Literatur zur Softwaretechnik:

- Sommerville (2004)
- Pressman (1997)
- Balzert (1997)
- Ludewig und Lichter (2006)

Termine

- montags, 8:30 – 10:00 Uhr, MZH 1090
- mittwochs, 14:00 s.t. – 15:30 Uhr, MZH 1090

Übungen und Ressourcen

Dozent:

- Erreichbar: TAB 2.57, Telefon 218-64481, koschke@tzi.de
- <http://www.informatik.uni-bremen.de/~koschke/>
- Sprechstunde nach Vereinbarung

Ressourcen:

- annotierte Folien unter http://www.informatik.uni-bremen.de/st/lehredetails.php?id=&lehre_id=412
- in der Vorlesung gezeigte und mit Tablet-PC beschriftete Folien in Stud.IP → registrieren!
- Videoaufzeichnungen aus dem Jahr 2007 unter <http://mlecture.uni-bremen.de/>
- News und annotierte Folien unter Stud.IP unter <http://elearning.uni-bremen.de>

Übungen:

- Übungen ca. alle zwei Wochen alternierend zur Vorlesung
- Übungsblatt im Netz

7 / 504

Scheinbedingungen

Anerkennung durch mündliche Prüfung:

- 30 minütige mündliche Prüfung über den Stoff der Vorlesung
- Übungsaufgaben bearbeiten lohnt sich

8 / 504

Lehrbücher I

Allgemeine Literatur zur Softwaretechnik

-  Sommerville (2004)
-  Pressman (1997)
-  Balzert (1997)
-  Ludewig und Lichter (2006)

Software-Metriken

-  Fenton und Pfleeger (1998)

Aufwand- und Kostenschätzung

-  Boehm u. a. (2000)
-  Poensgen und Bock (2005)

9 / 504

Lehrbücher II

Software-Entwicklungsprozesse

-  Beck (2000)
-  Kruchten (1998)
-  Balzert (2008)
-  Bunse und von Knethen (2002)
-  Pichler (2008)
-  auch: allgemeine Literatur über Softwaretechnik

Software-Prozessverbesserung

-  Siviuy u. a. (2007)
-  Kneuper (2006)

Komponentenbasierte Entwicklung

-  Szyperski u. a. (2002)

10 / 504

Lehrbücher III

Modellgetriebene Entwicklung

 Stahl u. a. (2007)

Software-Architektur

 Bass u. a. (2003)

 Hofmeister u. a. (2000)

Entwurfsmuster

 Gamma u. a. (2003)

 Buschmann u. a. (1996)

 Schmidt u. a. (2000)

Software-Produktlinien

 Clements und Northrop (2001)

11 / 504

Lehrbücher IV

Empirisches Software-Engineering

 Endres und Rombach (2003)

 Yin (2003)

 Prechelt (2001)

12 / 504

Entwicklungsprozesse

Entwicklungsprozesse

Lernziele
Wasserfallmodell
V-Modell
Testgetriebene Entwicklung
Inkrementelle Entwicklung
Spiralmodell
Rational Unified Process
Cleanroom Development
Agile Methoden
Extreme Programming (XP)
Scrum
Agile versus traditionelle Vorgehensweisen
Capability Maturity Model
Persönlicher Softwareprozess
Wiederholungsfragen
Weiterführende Literatur

13 / 504

Lernziele

- verschiedene Software-Entwicklungsprozessmodelle kennen
- Vor- und Nachteile/Anwendbarkeit abwägen können
- die Besonderheit von Prozessen der Software-Entwicklung kennen

14 / 504

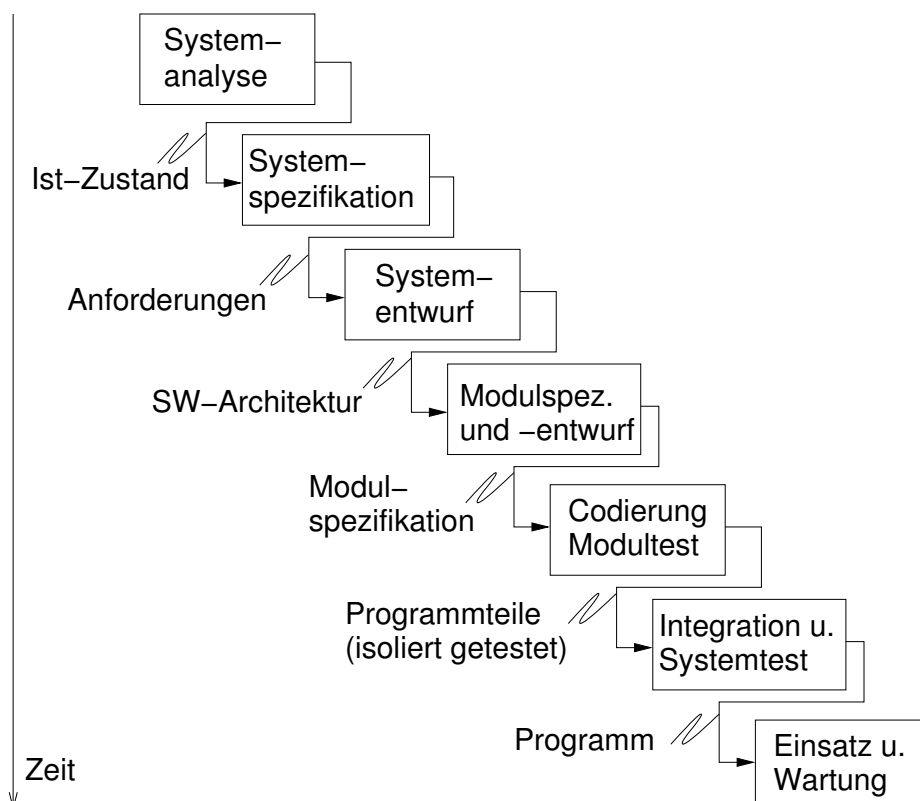
Entwicklungsprozesse

Grundlagen für guten Prozess:

- Wohldefiniertheit
 - sonst Falsch-, Nicht-, Mehrarbeit
 - sonst Information nicht verfügbar oder unverständlich
- Quantitatives Verstehen
 - objektive Grundlage für Verbesserungen
- Kontinuierliche Änderung
 - sonst Prozess schnell überholt
- Angemessenheit
 - Prozess muss dem zu lösenden Problem angemessen sein
 - d.h. effektiv und effizient sein

15 / 504

Striktes Wasserfallmodell nach Royce (1970)



16 / 504

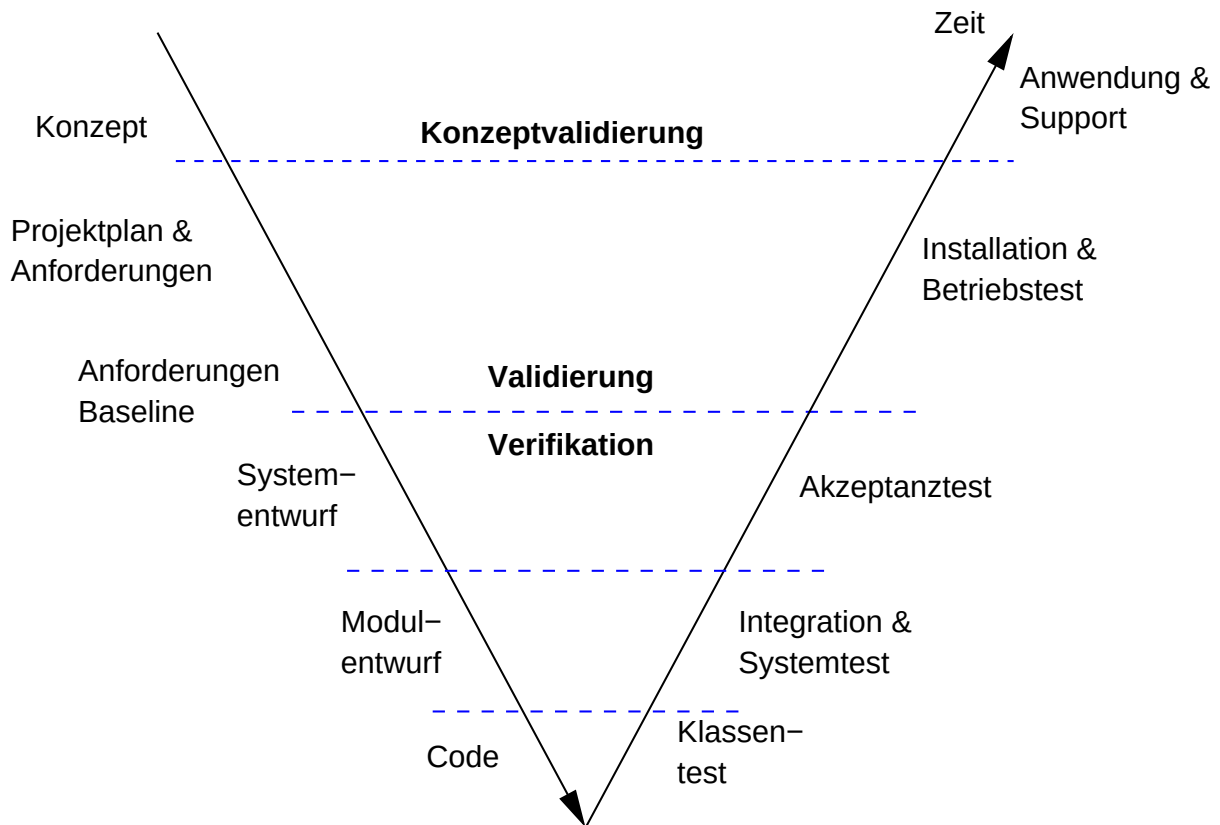
Hier sehen wir als Beispiel das strikte Wasserfallmodell. Es enthält alle Aktivitäten in streng sequenzieller Folge. Dieses Modell eignet sich, wenn die Arbeitsschritte deutlich voneinander getrennt werden können. Alle Risiken müssen vor Projektbeginn ausgeschlossen werden können. Es ist geeignet, wenn die Mitarbeiteranzahl klein ist, da alle Mitarbeiter gleichzeitig an einem Arbeitsschritt arbeiten können. Dieses Modell ist für die allermeisten Aufgabenstellung jedoch unrealistisch. Es setzt voraus, dass jede Aktivität auf Anhieb erfolgreich abgeschlossen werden kann. Allerdings werden z.B. die Anforderungen oft auch noch in späten Phasen des Projekts geändert bzw. erst dann überhaupt erst richtig verstanden. Dann müssen frühe Aktivitäten wiederholt werden.

Striktes Wasserfallmodell Royce (1970)

Eigenschaften dieses Modells:

- dokumenten-getrieben: jede Aktivität erzeugt Dokument
- streng sequenzielle Aktivitäten
- + klar organisierter Ablauf
- + relativ einfache Führung
- + hohe Qualität der Dokumentation
- Entwicklung bildet langen Tunnel
- 90%-fertig-Syndrom
- Spezifikationsmängel lassen sich kaum frühzeitig erkennen
- Entwicklung beim Kunden wird ignoriert

V-Modell von Boehm (1979)



18 / 504

Das V-Modell ist kein eigenständiges Prozessmodell im eigentlichen Sinn. Die Produkte im V-Modell werden wie im Wasserfallmodell streng sequenziell erstellt. Es fügt dem Wasserfallmodell lediglich eine zusätzliche strukturelle Sicht hinzu, nämlich dass für jedes zu erstellende Dokument ein entsprechender Test existieren und durchgeführt werden soll.

Das V-Modell sieht sowohl Verifikationen als auch Validierungen vor. Validierung: Es wird das richtige Produkt erstellt. Verifikation: Das Produkt wird richtig erstellt.

Eine weitere Konsequenz des V-Modells für die konkrete Projektplanung ist, dass man die Test- und Validierungsaktivitäten früher als die tatsächliche Durchführung der Aktivität vorbereiten kann. Beispielsweise kann der Akzeptanztest vorbereitet werden, sobald man die Anforderungen kennt. Auf diese Weise können Aktivitäten bei einem Projektteam parallelisiert werden: Der Tester kann Testfälle für den Akzeptanztest entwickeln, auch wenn noch keine Implementierung existiert.

V-Modell von Boehm (1979)

Eigenschaften:

- entspricht Wasserfallmodell
- + betont Qualitätssicherung
- + frühe Vorbereitung von Validierung und Verifikation
 - zusätzliche Parallelisierung
 - Fehler/Mängel werden früher entdeckt
- alle sonstigen Nachteile des Wasserfallmodells

19 / 504

Testgetriebene Entwicklung (Sneed 2004)

Kennzeichen testgetriebener Entwicklung:

Testfälle ...

- werden früh aus Anwendungsfällen abgeleitet
- dienen als Baseline
- treiben den Entwurf
- treiben die Kodierung

Test-Teams treiben die Entwicklung, statt von Entwicklern getrieben zu werden

20 / 504

Testgetriebene Entwicklung (Sneed 2004)

Anwendungs- und Testfälle

- Anwendungsfälle beschreiben Anforderungen
- sind jedoch oft nicht detailliert genug, um erwartetes Verhalten vollständig zu spezifizieren
- Testfälle können Anwendungsfälle hier komplementieren
- zu jedem Anwendungsfall sollte es mindestens einen Testfall geben
- Testfälle können zur Kommunikation zwischen Entwickler und Kunden/Anwender dienen

21 / 504

Testgetriebene Entwicklung (Sneed 2004)

Testfälle dienen als Baseline:

- definieren die Vorbedingungen einer Produktfunktion
- spezifizieren die Argumente einer Produktfunktion
- spezifizieren das Verhalten einer Produktfunktion (die Nachbedingungen)

22 / 504

Testgetriebene Entwicklung (Sneed 2004)

Testfälle treiben den Entwurf:

- Testfall ist ein Pfad durch die Software-Architektur
- Testfälle verknüpfen Anforderungsspezifikation und Architekturkomponenten
- Testfälle können zur Studie der zu erwartenden Performanz und anderer Systemattribute zur Entwurfszeit verwendet werden

23 / 504

Testgetriebene Entwicklung (Sneed 2004)

Testfälle treiben die Kodierung:

- vor Implementierung einer Klasse werden erst Testtreiber entwickelt
- Testtreiber enthält mindestens einen Test für jede nichttriviale Methode
- Testfall hilft, die richtige Schnittstelle zu definieren
- Testfall zwingt Entwickler, über das zu erwartende Resultat nachzudenken
- Testfälle spezifizieren die Methoden zumindest partiell

24 / 504

Testgetriebene Entwicklung (Sneed 2004)

Tester treiben die Entwicklung:

- Tester sind verantwortlich für die Auslieferung des Produkts
- Tester legen Kriterien für die Auslieferbarkeit fest
- Entwickler sind Lieferanten für die Tester
- Softwareentwicklung ist eine Versorgungskette; das Ende zieht, anstatt gedrückt zu werden

25 / 504

Bewusstseinssebenen

- bewusstes Wissen (20-30%)
 - Wissen, über das man sich im Klaren ist oder das in seiner vollen Bedeutung klar erkannt wird
- unbewusstes Wissen ($\leq 40\%$)
 - Wissen, das sich dem Bewusstsein im Moment nicht darbietet, aber dennoch handlungsbestimmend ist, und potenziell aufgerufen werden kann
- unterbewusstes Wissen
 - unbekannte Wünsche, die erst von außen herangetragen werden müssen, um als Anforderungen erkannt zu werden

26 / 504

- bewusst: will mit meinem Handy telefonieren
- unbewusst: Tastatur und Display sollen auf derselben Seite meines Handys sein
- unterbewusst: ich will SMS verschicken können

Basisfaktoren

- Minimalanforderungen
- Mangel führt zu massiver Unzufriedenheit
- mehr als Zufriedenheit ist nicht möglich

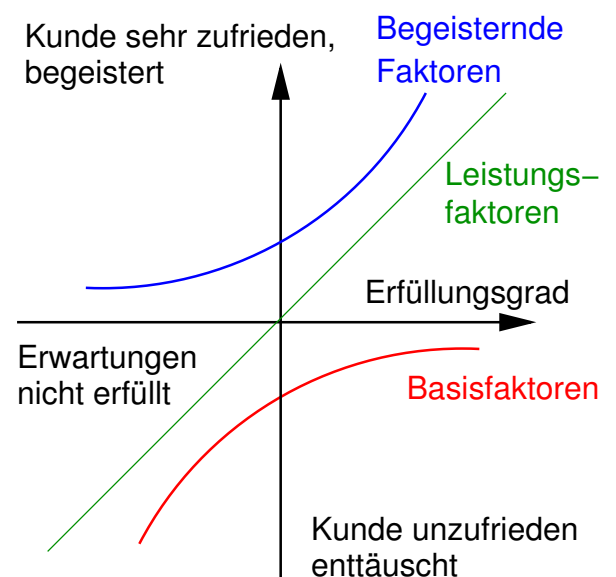
Leistungsfaktoren

- bewusst verlangte Sonderausstattung
- bei Erfüllung: Kundenzufriedenheit
- sonst: Unzufriedenheit

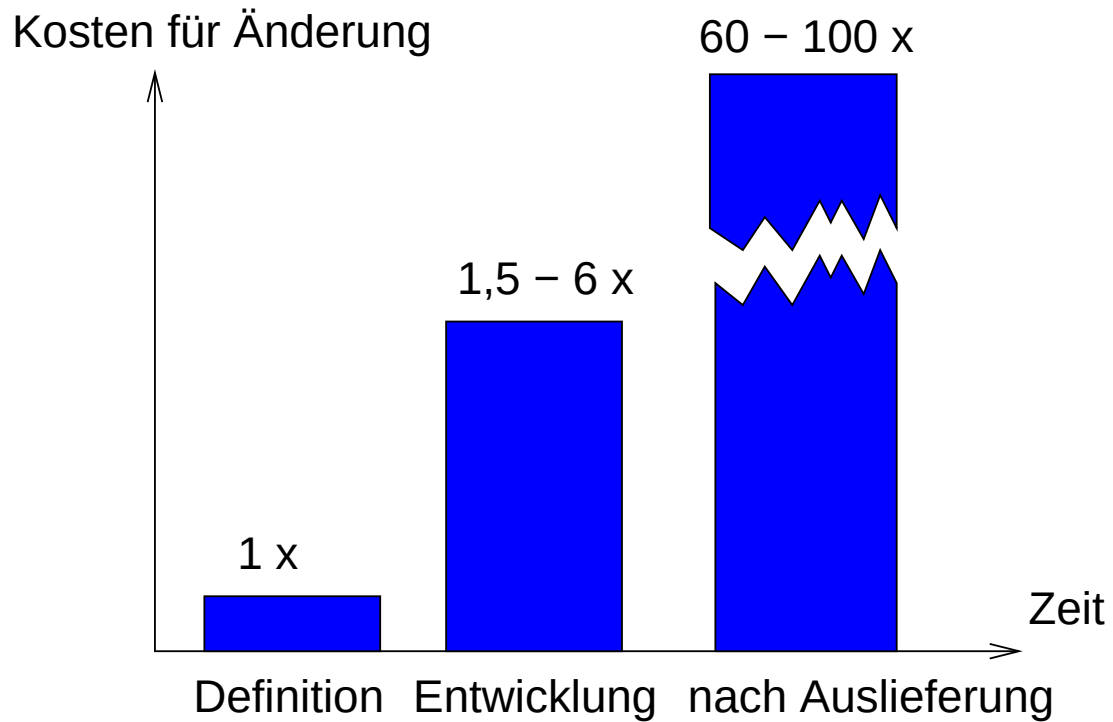
Begeisternde Faktoren

- unbewusste Wünsche, nützliche/angenehme Überraschungen
- steigern Zufriedenheit überproportional

Kano-Modell



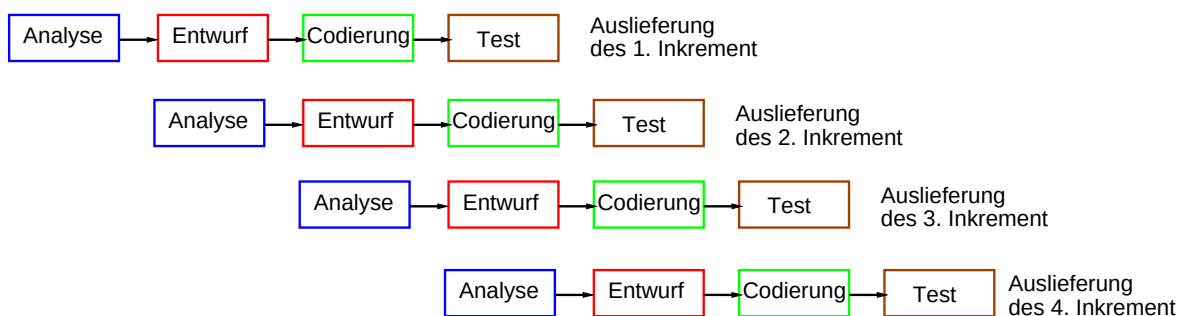
Kosten für Änderungen



Pressman (1997)

28 / 504

Inkrementelles Modell von Basili und Turner (1975)



29 / 504

Das Wasserfallmodell geht davon aus, dass alle Anforderungen zu Beginn erfasst werden können und diese während des Projekts stabil bleiben. Dies ist in der Praxis selten der Fall.

Im Laufe des Projekts gewinnen die Entwickler ein besseres Verständnis der Anwendungsdomäne und entdecken Irrtümer in ihren ursprünglichen – oft nur impliziten – Annahmen. Ähnlich wird auch der Kunde sich oft erst im Laufe des Projekts im Klaren, was er eigentlich erwarten kann und will. Auch die Rahmenbedingungen, unter denen das Projekt startete, können sich ändern (z.B. Gesetze oder Normen).

Die Anforderungen sind also selten stabil. Damit wird das Wasserfallmodell in seiner strikten Auslegung fragwürdig. Allerdings hat auch schon der Erfinder des Wasserfallmodells, Royce, empfohlen, das System zweimal zu entwickeln. Das erste Mal, um überhaupt erst Anforderungen und mögliche Lösungen auszuloten. Das zweite Mal, um ein adäquates und qualitativ hochwertiges Produkt zu erstellen.

Das inkrementelle Modell führt diesen Gedanken fort. Anstatt auf die Fertigstellung der gesamten Anforderungen zu warten, werden – sobald eine ausreichende Anzahl von Kernanforderungen beschrieben ist – für diese ein Entwurf und die Implementierung gestattet. Je mehr Anforderungen hinzukommen, desto mehr zusätzliche Inkremente werden gestartet.

Das inkrementelle Modell ist gut geeignet, wenn Kunde die Anforderungen noch nicht vollständig überblickt bzw. sich der Möglichkeiten zur Realisierung der Anforderungen nicht bewusst ist und deshalb die Anforderungen nicht formulieren kann.

Es ist mit diesem Modell möglich, Teile des Systems bereits vor Fertigstellung des gesamten Systems beim Kunden einzuführen (z.B. ein oder mehrere Subsysteme). Mit diesen Teilen kann der Kunde eine eingeschränkte Anzahl an Anforderungen realisieren. Die Zeitspanne zwischen Auftragsvergabe und Einsatz von zumindest Systemteilen wird somit geringer.

Federt die Gefahr des Wasserfallmodells ab, am Ende mit leeren Händen dazustehen. Sind wenigstens die ersten Inkremente erfolgreich entstanden, hat der Kunde zumindest ein partielles System.

Während der Entwicklung kann noch auf Änderungen reagiert werden.

Das Modell steht und fällt mit der Möglichkeit, Kernanforderungen zu identifizieren und ausreichend zu spezifizieren. Die initiale Software-Architektur muss für alle Anforderungen – Kernanforderungen wie alle anderen, die im Laufe des Projekts formuliert werden – tragfähig sein; ansonsten ist ein hoher Restrukturierungsaufwand notwendig. Darum sollten am Anfang alle Anforderungen bekannt sein, die sich maßgebend auf die Architektur auswirken.

Beispiel für Textverarbeitung

Iterationen:

- ① grundlegende Funktionalität
 - Datei-Management, Editor, Textausgabe
- ② erweiterte Funktionalität
 - Style-Files, Bearbeitung mathematischer Formeln, Einbinden von Graphiken
- ③ zusätzliche Funktionalität
 - Rechtschreibprüfung, Grammatiküberprüfung, Überarbeitungsmodus
- ④ ergänzende Funktionalität
 - Tabellenkalkulation, Geschäftsgraphiken, E-Mail, Web-Browser, Scanner-Anbindung, Flipper

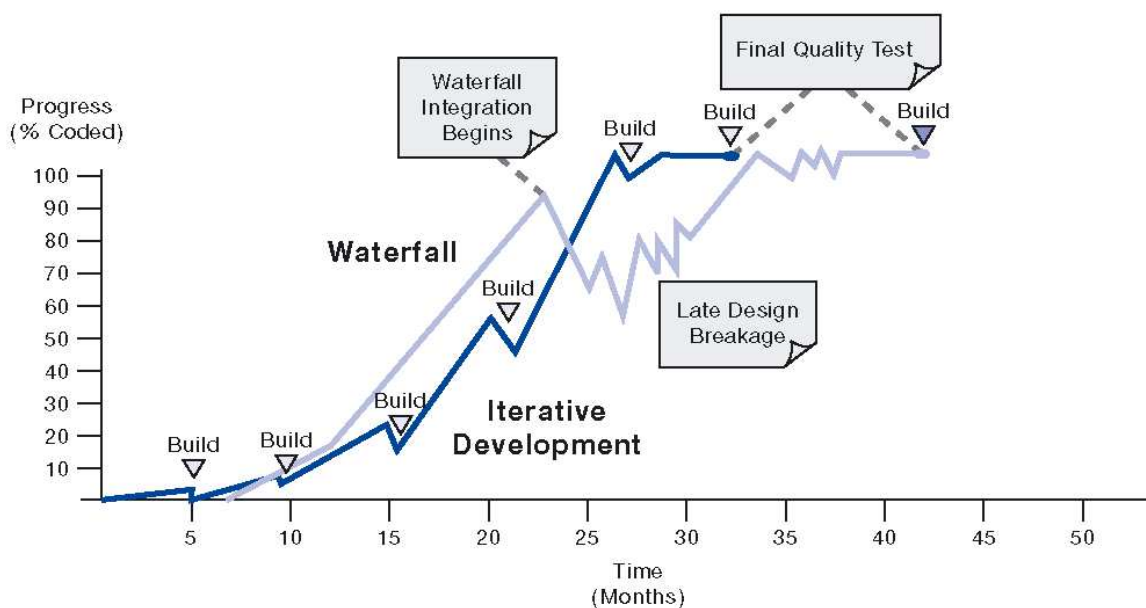
Inkrementelles Modell von Basili und Turner (1975)

Eigenschaften:

- Wartung wird als Erstellung einer neuen Version des bestehenden Produkts betrachtet.
- + Entwicklung erfolgt stufenweise
 - brauchbare Teillösungen in kurzen Abständen
- + Lernen durch Entwicklung und Verwendung des Systems.
- + Gut geeignet, wenn Kunde Anforderungen noch nicht vollständig überblickt oder formulieren kann.
 - *"I know it when I see it"*
- Kernanforderungen und Architekturvision müssen vorhanden sein.
- Entwicklung ist durch existierenden Code eingeschränkt.

31 / 504

Vergleich inkrementelles Modell und Wasserfallmodell



32 / 504

Spiralmodell von Boehm (1988)

Mehrere Iterationen der folgenden Schritte:

- ① **Bestimmung der Ziele** und Produkte des Durchlaufs; Berücksichtigung von Alternativen (z.B. Entwurfsvarianten) und Restriktionen (z.B. Zeitplan)
- ② **Bewertung der Risiken** für alle Alternativen; Entwicklung von Lösungsstrategien zur Beseitigung der Ursachen
- ③ **Arbeitsschritte durchführen**, um Produkt zu erstellen
- ④ Review der Ergebnisse und **Planung** der nächsten Iteration

Das Spiralmodell kann für sehr große und komplexe Projekte verwendet werden, da es der Komplexität durch das risikogesteuerte Vorgehen Rechnung trägt. Die Anzahl der Durchläufe ergibt sich erst während des Projekts und wird durch die auftretenden Risiken bestimmt. Dies hat zur Folge, dass zu Beginn des Projekts ein Zeit- und Kostenplan nur schwer zu erstellen ist. Die Risikoanalyse kann nur durch erfahrene Projektleiter durchgeführt werden. Bei zu zaghaftem Vorgehen kann sich das Projekt unnötigerweise verlängern, was zu erhöhten Kosten führt. Zu schnelles Vorgehen kann Risiken vernachlässigen und zu Problemen in folgenden Durchläufen führen.

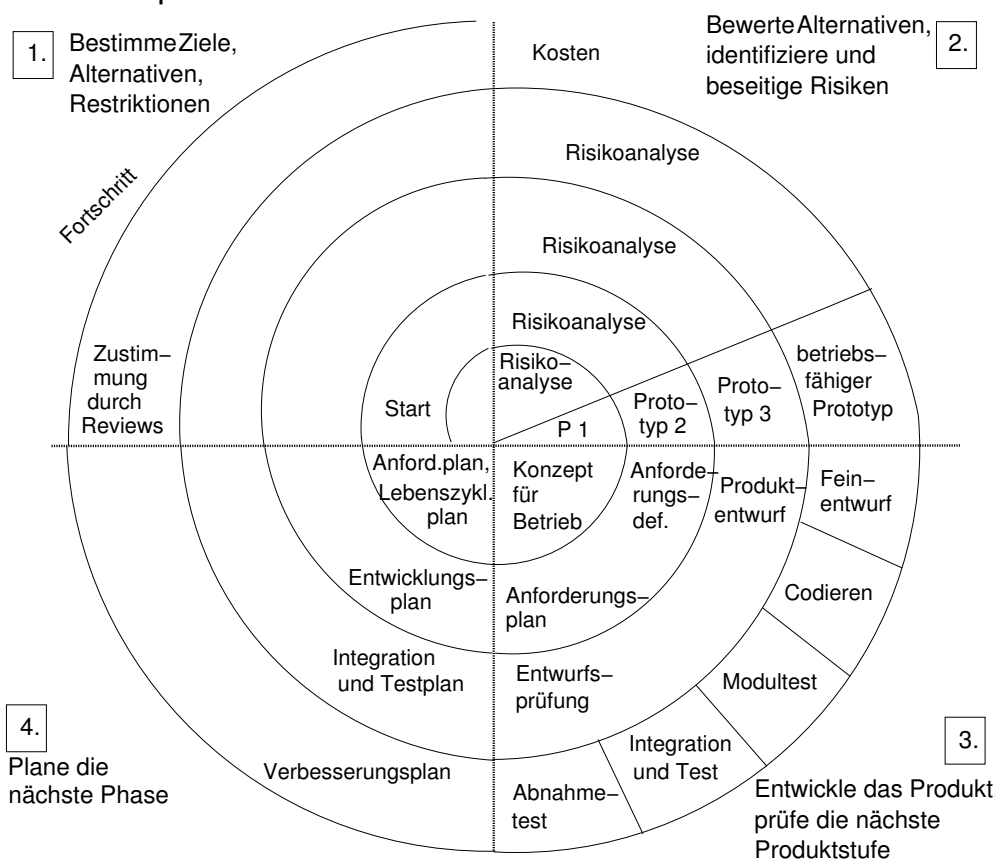
Beispiel eines Spiralmodells

(Generische) Risiken:

- Ist das Konzept schlüssig? Kann es aufgehen?
- Was sind die genauen Anforderungen?
- Wie sieht ein geeigneter Entwurf aus?

35 / 504

Beispiel eines Spiralmodells mit vier Durchläufen



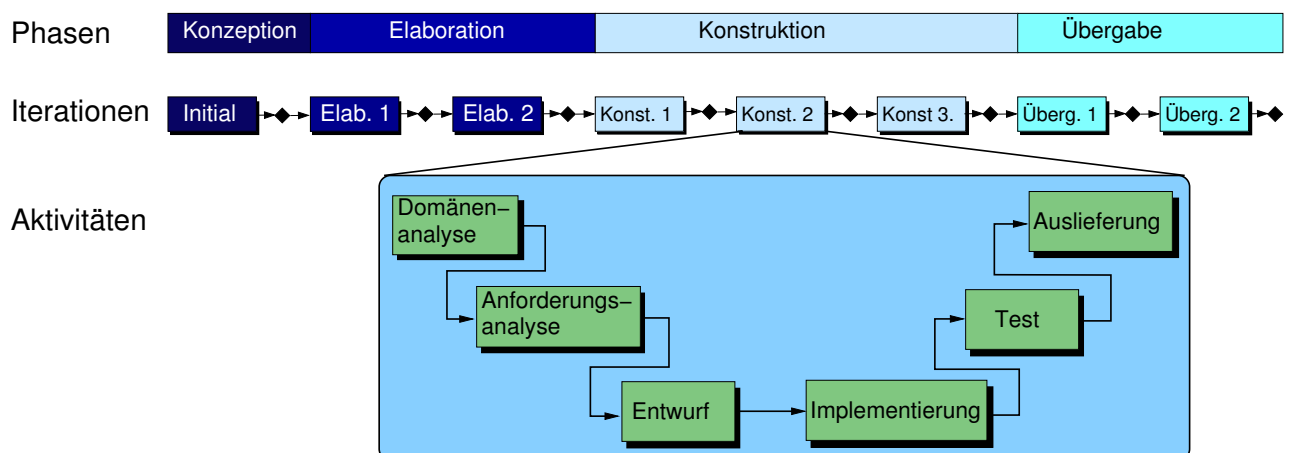
36 / 504

Bewertung Spiralmodell

- Meta-Modell: Iterationen können beliebigen Modellen folgen
- + bei unübersichtlichen Ausgangslagen wird die Entwicklung in einzelne Schritte zerlegt, die jeweils unter den gegebenen Bedingungen das optimale Teilziel verfolgen
- schwierige Planung (was jedoch dem Problem inhärent ist)
- setzt große Flexibilität in der Organisation und beim Kunden voraus

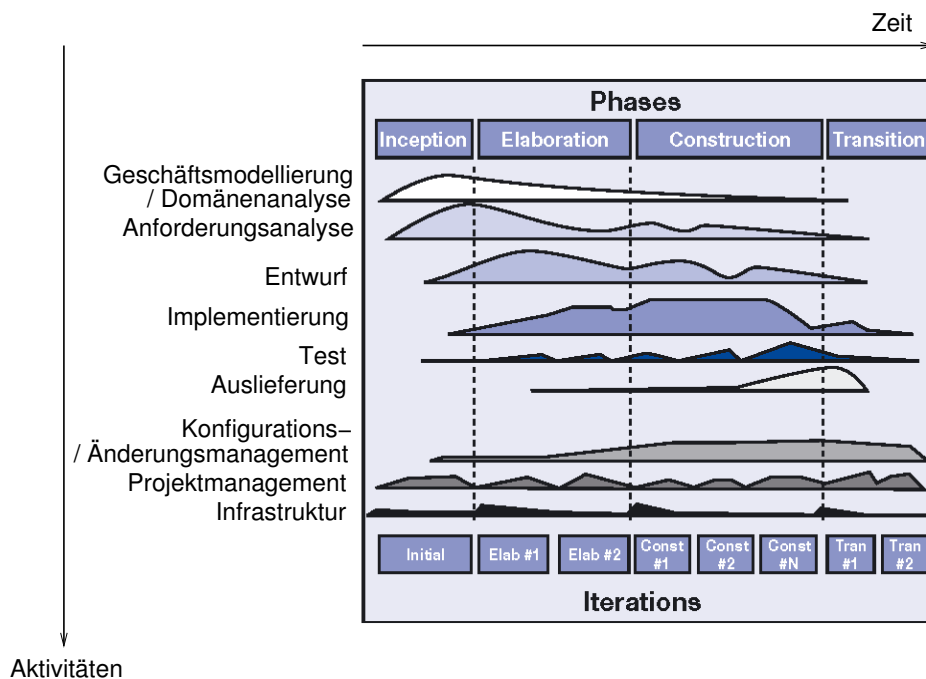
37 / 504

Rational Unified Process (RUP) nach Gornik (2001)



38 / 504

Rational Unified Process (RUP) nach Gornik (2001)



39 / 504

Der Rational Unified Process (RUP) – als Konkretisierung des Unified Process der Firma Rational – ist ein weiteres inkrementelles Modell.

Der Prozess insgesamt gliedert sich in vier Phasen mit einer unterschiedlichen Anzahl von Iterationen. Das Produkt jeder Phase unterscheidet sich von den Produkten anderer Phasen.

Die Konzeptionsphase (Inception) erarbeitet die Anforderungen. Die Elaborationsphase (Elaboration) erstellt einen Entwurf. Die Konstruktionsphase (Construction) erstellt das implementierte System. Die Übergabephase (Transition) führt das System beim Kunden ein.

Jede Iteration gliedert sich in die Aktivitäten Geschäftsmodellierung, Anforderungsanalyse, Entwurf, Implementierung, Test und Auslieferung. Jede Iteration wird durch ein formales Review abgeschlossen.

Die Ausprägung der einzelnen Aktivitäten ist phasenabhängig. In der Konzeptphase z.B. dient eine Implementierung lediglich einem Konzeptbeweis (Machbarkeit kritischer Anforderungen) oder einer Demonstration (ein Benutzerschnittstellenprototyp). Es genügt hierfür eine weniger aufwändige, prototypische Implementierung (der Prototyp sollte anschließend weggeworfen werden!).

Der Aufwand jeder Aktivität variiert also in den Phasen. Dies wird durch die Kurven im Schaubild veranschaulicht. Die Geschäftsmodellierung beispielsweise erzeugt in der Konzeptionsphase naturgemäß einen hohen Aufwand, hat in der Übergabephase aber keine Bedeutung mehr. Alle Flächen gemeinsam ergeben den Gesamtaufwand des Projekts. Die Summe der Flächen pro Spalte ist der Aufwand pro Iteration. Die Summe der Flächen pro Zeile ist der Aufwand pro Aktivität.

Die Hauptaktivitäten sind Geschäftsmodellierung (optional), Anforderungsanalyse, Entwurf, Implementierung, Test, Auslieferung. Die Geschäftsmodellierung dient dazu, eine gemeinsame Sprache für die unterschiedlichen Gruppen Softwareentwickler und Betriebswirte zu finden und die Software-Modelle auf die zugrunde liegenden Geschäftsmodelle zurückzuführen. Die Geschäftsprozesse werden durch so genannte Geschäfts-Anwendungsfälle dokumentiert. Sie zielen darauf ab, ein gemeinsames Verständnis, welche Geschäftsprozesse in der Organisation unterstützt werden sollen, aller Beteiligten zu erreichen. Die Geschäfts-Anwendungsfälle werden analysiert, um zu verstehen, wie die Geschäftsprozesse unterstützt werden sollen.

RUP: Konzeptionsphase (Inception)

Ziel: "Business-Case" erstellen und Projektgegenstand abgrenzen.

Resultate:

- Vision der Hauptanforderungen, Schlüsselfeatures und wesentliche Einschränkungen
- initiale Anwendungsfälle (10-20% vollständig)
- Glossar oder auch Domänenmodell
- initialer Business-Case: Geschäftskontext, Erfolgskriterien (Schätzung des erzielten Gewinns, Marktanalyse etc.) und Finanzvorschau
- initiale Risikobetrachtung
- Projektplan mit Phasen und Iterationen
- Business-Modell falls notwendig
- ein oder mehrere Prototypen

40 / 504

Begleitende Aktivitäten sind das Konfigurations- und Änderungsmanagement und das Projektmanagement. Ihr Aufwand ist in allen Phasen mehr oder minder gleich. Der Aufwand für das Konfigurations- und Änderungsmanagement zeigt leichte Peaks im Übergang von einer Phase zur anderen, wenn die Konfigurationen fest gezurrt werden und zum Teil nachgearbeitet werden muss.

Dem Glossar kommt eine ganz besondere Bedeutung zu. Es erklärt die Begriffe der Anwendungsdomäne.

Software-Entwickler sind Spezialisten für die Entwicklung von Software, aber Laien in sehr vielen ihrer Anwendungsdomänen. Darüber hinaus verwenden auch Kunden oft die Begriffe uneinheitlich bzw. geläufige Worte mit einer ganz speziellen Bedeutung in ihrem Kontext. Mißverständnisse zwischen Kunden und Softwareentwickler sind sehr häufig und können zu teuren Fehlentwicklungen führen.

Eine Marssonde, bei deren Entwicklung europäische und amerikanische Organisationen mitwirkten, verfehlte ihr Ziel, weil den Organisationen nicht bewusst war, dass sie unterschiedliche metrische Systeme für ihre Software zugrunde legten. Die einen interpretierten einen Wert in Zentimetern, die anderen in Zoll (Inch).

Im Glossar beschreibt der Software-Entwickler, was die Begriffe des Kunden bedeuten, ebenso wie seine eigenen speziellen Begriffe. Der Kunde begutachtet das Glossar. Damit definieren beide Parteien ihr Vokabular.

Mißverständnisse sollen so minimiert werden.

Das Domänenmodell (oft auch konzeptuelles Modell genannt) beschreibt die Begriffe/Objekte der Anwendungsdomäne und ihre Relationen.

Eine Reihe der genannten Punkte wird sicherlich in Zusammenarbeit mit Betriebswirten ausgearbeitet.

Softwareentwickler haben eine Liebe zur Technologie, übersehen jedoch leider oft die Wirtschaftlichkeit ihrer Ideen. Mit ihr steht und fällt jedoch das Projekt.

Die Erstellung von Prototypen hat das Ziel, mögliche technologische Risiken auszuschließen, dem Kunden ein konkretes Bild der möglichen Anwendung zu vermitteln (Benutzerschnittstellenprototyp), Anforderungen zu konkretisieren ("I know it when I see it") und die Machbarkeit bestimmter Anforderungen zu demonstrieren.

Das Business-Modell erläutert, wie das System eingesetzt wird, um damit Profite zu erzielen.

RUP: Elaborationsphase (Elaboration)

Ziel: Verständnis der Anwendungsdomäne, tragfähige Software-Architektur, Projektplan, Eliminierung der Risiken

- Anwendungsfallmodell (mind. 80% fertig)
 - alle Anwendungsfälle und Aktoren sind identifiziert,
 - die meisten Anwendungsfallbeschreibungen wurden entwickelt
- zusätzliche nichtfunktionale Anforderungen und Anforderungen, die nicht mit einem spezifischen Anwendungsfall assoziiert sind
- Beschreibung der Software-Architektur
- ausführbarer Architekturprototyp

41 / 504

Die Elaborationsphase ist die kritischste Phase. Hier entscheidet sich, ob das System tatsächlich gebaut wird. Der Engineering-Anteil ist weitgehend erbracht. Bis dahin halten sich die Kosten noch in Grenzen. Nun schließen sich die teure Konstruktions- und Übergabephase an.

Die Aktivitäten der Elaborationsphase stellen sicher, dass die Software-Architektur, die Anforderungen und Pläne hinreichend stabil sind (mögliche Änderungen sind antizipiert, völlig ausschließen lassen sie sich meist nicht) und Risiken sind ausreichend betrachtet, so dass Kosten und Zeitplan zuverlässig geschätzt werden können. Ab hier sollte man sich auf eine Projektdurchführung mit festem Preis einlassen können.

In der Elaborationsphase wird ein ausführbarer Architekturprototyp in ein oder mehr Iterationen erstellt. Die Anzahl der Iterationen hängt vom Scope, der Größe, der Risiken und dem Grad des Unbekannten des Projekts ab.

Zumindest die kritischen Anwendungsfälle sollten hierfür einbezogen werden, da sie typischerweise die größten technischen Risiken aufwerfen. Ein evolutionärer Prototyp (d.h. einer der schrittweise ausgebaut wird) kann durchaus verwendet werden. Man sollte jedoch auch einige explorative Wegwerfprototypen in Erwägung ziehen, um spezifische Risiken wie z.B. Entwurfs- oder Anforderungskompromisse auszuloten. Sie dienen auch als Machbarkeitsstudien und Demonstrationen.

RUP: Elaborationsphase (Elaboration); Fortsetzung

- überarbeitete Liste der Risiken und überarbeiteter Business-Case
- Plan für das gesamte Projekt sowie grober Plan für die Iterationen und Meilensteine
- ein vorläufiges Benutzerhandbuch (optional)

42 / 504

Die Erstellung des Benutzerhandbuchs kann bereits frühzeitig beginnen. Es ist konkreter als die Anforderungsspezifikation und abstrakter als die Implementierung. Somit kann es als Brücke von den Anforderungen zur Implementierung benutzt werden.

Das vorläufige Handbuch dient sowohl als Spezifikation für die Implementierung und den Test als auch für die intensivere Auseinandersetzung mit der Benutzerführung (Beispiel: Was orthogonal und einfach zu beschreiben ist, kann oft auch orthogonal und einfach implementiert werden). Überlegungen zur Benutzerführung sollten frühzeitig gemacht werden, weil Änderungen größere Restrukturierungen nach sich ziehen können.

RUP: Konstruktionsphase (Construction)

Ziel: Fertigstellung, Integration und Test aller Komponenten; auslieferbares Produkt.

- Software-Produkt integriert in die entsprechende Plattform
- Benutzerhandbuch
- Dokumentation des gegenwärtigen Releases

43 / 504

In der Konstruktionsphase werden alle übrigen Komponenten und Feature realisiert und in das Produkt integriert und intensiv getestet. Die Konstruktionsphase ist einem gewissen Sinne ein Herstellungsprozess, bei dem Wert auf das Management von Ressourcen und das Controlling gelegt wird, um Kosten, Zeitplan und Qualität zu optimieren. In diesem Sinne geht der Prozess nun von der intellektuellen Entwicklung zur Entwicklung auslieferbarer Produkte über.

Viele Projekte sind groß genug, um die Konstruktion zu parallelisieren. Die Parallelisierung kann die Verfügbarkeit auslieferbarer Releases beschleunigen. Andererseits kann sie auch die Komplexität der Ressourcenverwaltung und der Synchronisation der Arbeitsflüsse erhöhen.

Eine robuste Architektur und ein verständlicher Plan hängen stark zusammen. Deshalb ist eine kritische Qualität der Architektur die Einfachheit ihrer Konstruktion. Dies ist einer der Gründe, weshalb die ausgeglichene Entwicklung der Architektur und des Plans während der Elaborationsphase so sehr betont wird.

Das Resultat der Konstruktionsphase ist ein Produkt, das tatsächlich in die Hände des Benutzers übergehen kann.

RUP: Übergabephase (Transition)

Ziel: Produkt wird der Benutzergemeinde übergeben.

Hauptziele im Einzelnen:

- Benutzer sollten sich möglichst alleine zurecht finden.
- Beteiligte sind überzeugt, dass die Entwicklungs-Baselines vollständig und konsistent mit den Evaluationskriterien für die Vision sind.
- Erreichung der letzten Produkt-Baseline so schnell und kostengünstig wie möglich.

44 / 504

RUP: Übergabephase (Transition)

Typische Tätigkeiten:

- “Beta-Test”, um das neue System gegen die Benutzererwartungen zu validieren
- Parallele Verwendung mit einem Legacy-System, das durch das Produkt ersetzt werden soll
- Konversion aller notwendigen Daten (Dateien und Datenbanken)
- Schulung aller Benutzer und Administratoren
- Übergabe an Marketing, Vertrieb und Verkäufer

45 / 504

Wenn ein Produkt ausgeliefert wird, ergibt sich in der Regel schnell die Notwendigkeit neuer Releases, um Probleme zu beseitigen, Features zu realisieren, deren Implementierung verschoben werden musste, und Erweiterungen vorzunehmen.

Die Übergabephase beginnt, wenn eine Baseline reif genug ist, um beim Endbenutzer installiert werden zu können. Das erfordert typischerweise, dass zumindest eine nützliche Teilmenge des Systems mit einer akzeptablen Qualität fertig gestellt werden konnte und dass die Benutzerdokumentation vorhanden ist.

Meist fallen mehrere Iterationen in der Übergabephase an: Beta-Releases, allgemeine Releases, Bug-Fix- und Erweiterungsreleases. Hoher Aufwand wird in die Entwickler der benutzerorientierten Dokumentation, die Schulung von Benutzern, Unterstützung der Benutzer in ihren ersten Verwendungen des Produkts und Reaktion auf Benutzer-Feedback investiert.

Man sollte sich an diesem Punkt jedoch auf den Feinschliff, die Konfiguration, Installation und Usability-Aspekte beschränken. Gänzlich neue Erweiterungen sollten durch einen nachfolgenden separaten Entwicklungszyklus realisiert werden.

Die Übergabephase kann trivial sein (Software und Handbuch wird auf einen Server im Internet gelegt) oder auch sehr aufwändig und kompliziert (Ersetzung einer Flugaufsichts-Software).

Empfohlene Anzahl von Iterationen nach Kruchten (1998)

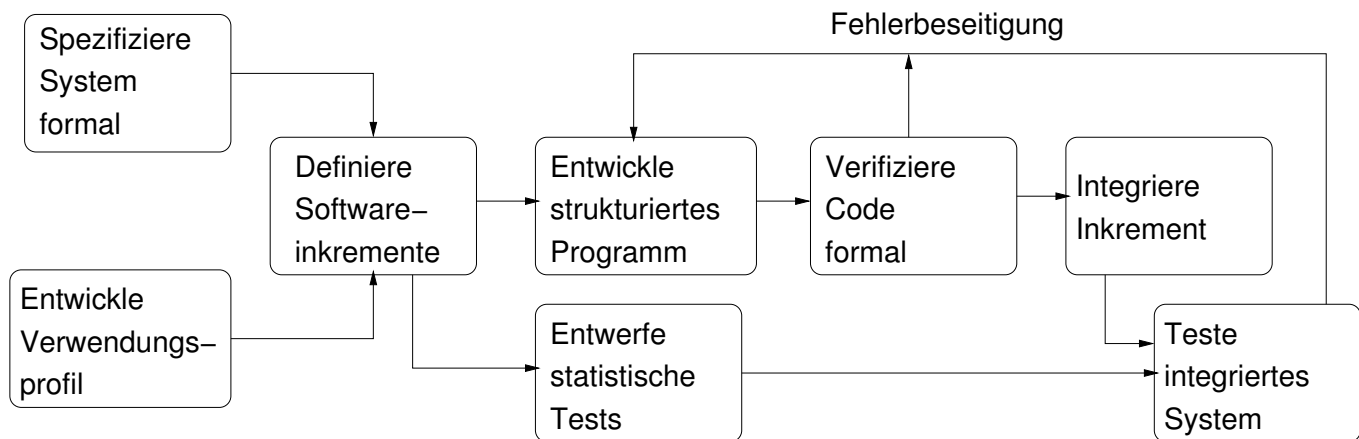
Komplexität	niedrig	normal	hoch
Konzeption	0	1	1
Elaboration	1	2	3
Konstruktion	1	2	3
Übergabe	1	1	2
Summe	3	6	9

Bewertung des RUPs

- Übernimmt vom Spiralmodell die Steuerung durch Risiken
- Konkretisiert die Aktivitäten (Spiralmodell ist ein Meta-Modell)
- Änderungen der Anforderungen sind leichter einzubeziehen als beim Wasserfallmodell
- Projekt-Team kann im Verlauf hinzulernen
- (Hauptsächlich) Konstruktionsphase kann inkrementell ausgestaltet werden

Iterativ ist nicht gleich inkrementell. Bei der iterativen Entwicklung werden Entwicklungsschritte wiederholt ausgeführt. Bei der inkrementellen Entwicklung geschieht dies auch, jedoch immer um eine neues Release auf Basis eines vorherigen zu bauen. Letzteres ist im Begriff Iteration nicht eingeschlossen.

Cleanroom Development (Mills u. a. 1987)



49 / 504

Cleanroom Development

Schlüsselstrategien

- Formale Spezifikation
- Inkrementelle Entwicklung
- Strukturierte Programmierung
- Statische Verifikation
- Statistisches Testen
 - basiert auf Verwendungsprofilen (die Verwendungsweise der Software in der Praxis)
 - die häufigsten (und kritischsten) Verwendungsarten werden verstärkt getestet

50 / 504

Statistisches Testen gleicht einem statistischen Experiment. Aus der formalen Spezifikation und den im Feld ermittelten Benutzungsprofilen werden geeignete Testfälle entwickelt. Mit Hilfe der Ergebnisse des Tests wird dann die Mean-Time-To-Failure (durchschnittliche Dauer bis zu einem Störfall) mit einer gewissen Wahrscheinlichkeit vorhergesagt.

Cleanroom Development

Gruppen:

- Spezifikationsteam:
 - verantwortlich für Entwicklung und Wartung der Systemspezifikation
 - erstellt kundenorientierte und formale Spezifikation
- Entwicklungsteam:
 - verantwortlich für Entwicklung und Verifikation der Software
 - Software wird nicht ausgeführt hierzu!
 - verwendet Code-Inspektion ergänzt durch Korrektheitsüberlegungen (nicht streng formal)
- Zertifizierungsteam:
 - verantwortlich für statistische Tests

Cleanroom Development

Erfahrungen Cobb und Mills (1990):

- weniger Fehler als bei traditioneller Entwicklung
- bei vergleichbaren Kosten

52 / 504

Agiles Manifest

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

— <http://agilemanifesto.org/>

53 / 504

Prinzipien der agilen Entwicklung

- Kundenzufriedenheit durch schnelle Auslieferung nutzbarer Software
- funktionsfähige Software wird häufig ausgeliefert (eher Wochen als Monate)
- funktionsfähige Software ist das Maß für Fortschritt
- auch späte Änderungen der Anforderungen sind willkommen
- enge tägliche Kooperation von Geschäftsleuten und Entwicklern
- Konversation von Angesicht zu Angesicht ist die beste Kommunikationsform (gemeinsamer Ort)
- Projekte bestehen aus Menschen, denen man vertrauen sollte
- kontinuierliches Streben nach technischer Exzellenz und gutem Design
- Einfachheit
- selbstorganisierte Teams
- kontinuierliche Anpassung an sich ändernde Bedingungen

— <http://www.agilemanifesto.org/principles.html>

55 / 504

Varianten der agilen Entwicklung

- Agile Unified Process (AUP)
- Dynamic Systems Development Method (DSDM)
- Essential Unified Process (EssUP)
- **Extreme Programming (XP)**
- Feature Driven Development (FDD)
- Open Unified Process (OpenUP)
- **Scrum**

Extreme Programming (Beck 2000)

Extreme Programming (XP) ist eine agile Methode für

- kleinere bis größere Entwicklerteams (max. 10-15 Personen),
- Probleme mit vagen Anforderungen
- und Projekte, bei denen ein Kundenrepräsentant stets greifbar ist.

<http://www.extremeprogramming.org/>

57 / 504

Extreme Programming (Beck 2000)

Anerkannte Prinzipien und Praktiken werden „extrem“ umgesetzt:

- Code-Reviews → permanente Reviews durch Pair-Programming
- Testen → ständiges Testen: Unit-Tests sowie Akzeptanztests durch den Kunden/Benutzer
- klare Struktur → jeder verbessert sie kontinuierlich durch Refactoring
- Einfachheit → stets die einfachste Struktur wählen, die die aktuellen Anforderungen erfüllt
- Integration → permanente Integration auch mehrmals am Tag
- Validierung:
 - Kunde/Benutzer ist stets involviert bei der Planung neuer Iterationen und verantwortlich für Akzeptanztest
 - kurze Iterationen → Dauer in Minuten und Stunden, nicht Wochen, Tage, Jahre

aber auch Auslassung anerkannter Prinzipien:

- Dokumentation: mündliche Überlieferung, Tests und Quellcode
- Planung: sehr begrenzter Horizont

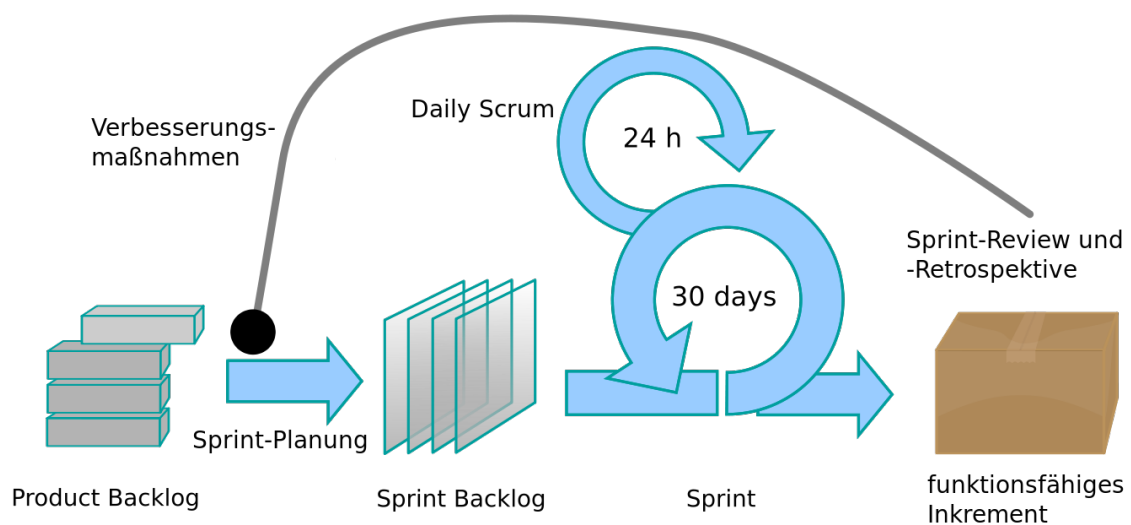
58 / 504

Weitere XP-Charakteristika

- Kunde vor Ort
- eine Metapher statt einer Architekturbeschreibung
- 40-Stundenwoche
- Code ist kollektives Eigentum
- Kodierungsstandards

59 / 504

Scrum-Prozess



61 / 504

Scrum-Rollen (Pichler 2008)

Product Owner:

- vertritt Endkundenbedürfnisse
- vereint Produkt- und Projektmanagementaufgaben
- fest integriert in Entwicklung

Aufgaben:

- Anforderungsbeschreibung und -management
- Releasemanagement und Return on Investment
- enge Zusammenarbeit mit dem Team
- Stakeholder-Management

Scrum-Rollen (Pichler 2008)

Team:

- entscheidet selbst, wieviele Anforderungen in einem Inkrement umgesetzt werden
- legt Arbeitsschritte und Arbeitsorganisation selbst fest
- agiert autonom
- interdisziplinär besetzt
- selbstorganisiert
- klein
- Vollzeitmitglieder
- Arbeitsplätze in unmittelbarer Nähe

63 / 504

Scrum-Rollen (Pichler 2008)

Scrum-Master (vom Team bestimmt):

Aufgaben

- etabliert Scrum
- unterstützt Team
- stellt Zusammenarbeit von Team und Product Owner sicher
- beseitigt Hindernisse
- verbessert Entwicklungspraktiken
- führt durch dienen

Fähigkeiten

- Moderation
- Coaching
- Erfahrung in Softwareentwicklung

64 / 504

Scrum: Product Backlog

Product Backlog enthält

- alle bekannten funktionalen und nichtfunktionalen Anforderungen
- weitere Arbeitsergebnisse (z.B. Aufsetzen der Test- und Entwicklungsumgebung)

keine Aktivitäten!

wird vom Product Owner gepflegt

65 / 504

Scrum: Product Backlog

Prio	Thema	Beschreibung	Akzeptanz	Aufwand
1	Kalender	Administrator kann zentrale Kalenderdatenbank anlegen	Installationsskript legt DB an	2
2	Kalender	Nutzer kann Termin eintragen	valider Termin wird eingetragen, invalider Termin wird zurückgewiesen	3
3	Kalender	Nutzer kann Termin löschen	gelöschter Termin ist entfernt; Löschung nur, wenn Rechte vorhanden	3
...	...	...	...	...

66 / 504

Das Product Backlog ist ein lebendes Dokument. Es wird nach jedem Sprint aktualisiert. Seine Einträge sind priorisiert. Sie weisen einen unterschiedlichen Detaillierungsgrad auf: Die im kommenden Sprint anvisierten Anforderungen sind genauer beschrieben. Der Aufwand jedes Eintrags ist abgeschätzt (Einheit: Punkte; siehe unten).

Vorlagen gibt es z.B. unter

http://epf.eclipse.org/wikis/scrum/Scrum/guidances/templates/burndown_chart_D182CF23.html

Scrum: Kostenschätzung

Schätzklausur und Planungspoker

Scrum: Kostenschätzung

Punkteskala (Fibonacci-Reihe):

0	kein Aufwand
1	sehr kleiner Aufwand
2	kleiner Aufwand = $2 \times$ sehr kleiner Aufwand
3	mittlerer Aufwand = sehr kleiner + kleiner Aufwand
5	großer Aufwand = kleiner + mittlerer Aufwand
8	sehr großer Aufwand = mittlerer + großer Aufwand
13	riesiger Aufwand = großer + sehr großer Aufwand

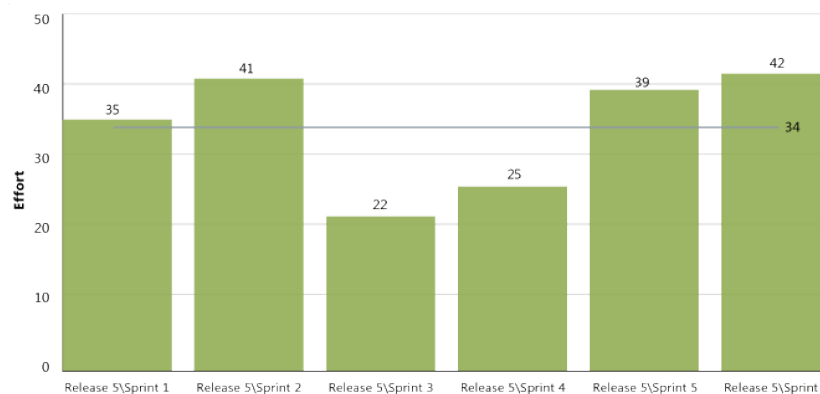
68 / 504

Scrum: Entwicklungsgeschwindigkeit

Punkte werden im Projektverlauf auf Echtzeit abgebildet

Entwicklungsgeschwindigkeit (Velocity) = Punkte / Sprint

Velocity Chart



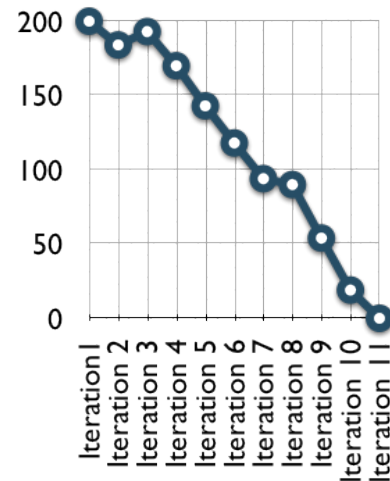
69 / 504

Scrum: Steuerungselemente

Sprint Burndown Chart



Release Burndown Chart



70 / 504

Agile versus weit voraus planende Prozessmodelle

Risiken agiler Methode:

- Skalierbarkeit, Kritikalität, Einfachheit des Entwurfs, Personalfluktuations, Personalfähigkeiten

Risiken weit voraus planender Prozessmodelle:

- Stabilität der Anforderungen, steter Wandel, Notwendigkeit schneller Resultate, Personalfähigkeiten

Generelle Risiken:

- Unsicherheiten bei der Technologie, unterschiedliche Interessengruppen, komplexe Systeme

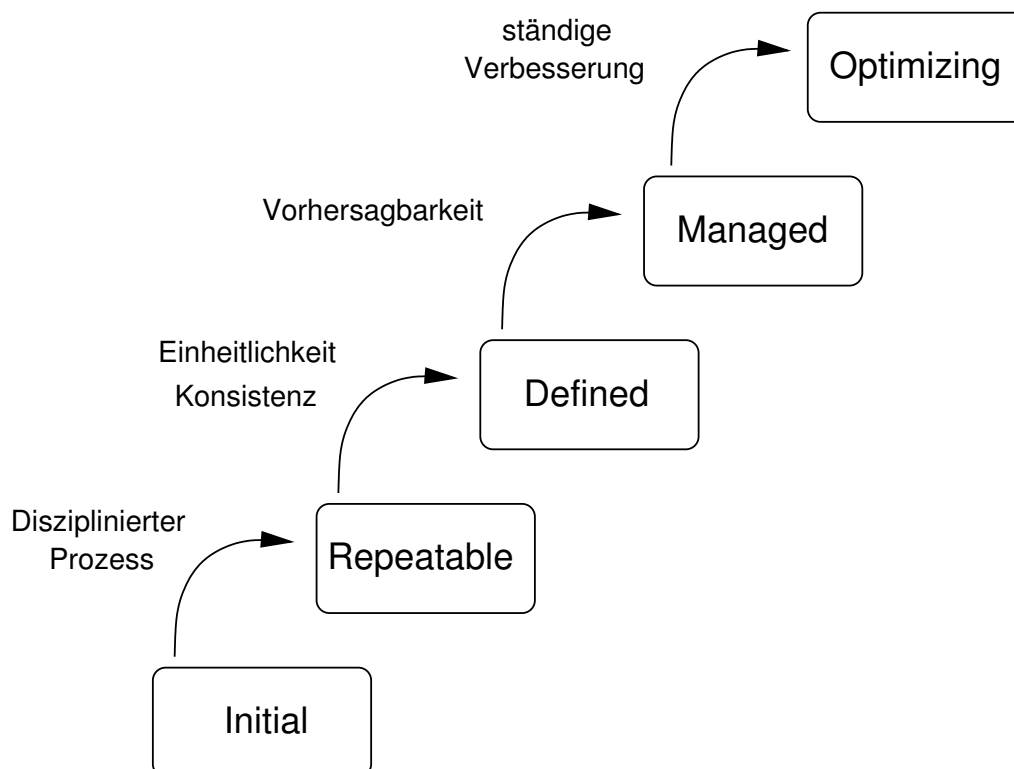
— Boehm und Turner (2003)

Capability Maturity Model

- Entwickelt vom SEI 1985-91 für DoD
- Beschreibt Stufen der Prozessreife
- Maßstab/Leitfaden für Verbesserungen
- Idee: besserer Prozess → besseres Produkt
- 5 Stufen (CMM Level 1-5)
- Definiert Schlüsselbereiche (Key Process Areas)
- Steigende Transparenz des Prozesses

72 / 504

Capability Maturity Model



73 / 504

CMM Level 1 – Initial

- Prozess meist völlig instabil
- Typisch: in Krisen nur noch Code & Fix
- Qualität und Fertigstellung unvorhersagbar
- Erfolge nur durch gute Leute und großen Einsatz

74 / 504

CMM Level 2 – Repeatable

- Projekterfolge nachvollziehbar und wiederholbar
- Projektplanung und -management basiert auf Erfahrung
- Key Process Areas:
 - Anforderungsverwaltung (u.a. Reviews)
 - Projektplanung (Zeitplanung, Risikomgmt., Prozess)
 - Projektverfolgung und -überblick (Transparenz)
 - Unterauftragsverwaltung
 - Qualitätssicherung (QS-Plan)
 - Konfigurationsverwaltung (Konsistenz, Änderungsverfolgung)

75 / 504

CMM Level 3 – Defined

- Organisationsweiter Prozess, wird für jedes Projekt angepasst
- Zuständig: Software Engineering Process Group
- Kosten, Zeitplan und Funktionalität im Griff
- Key Process Areas:
 - Organisationsweiter Prozess
 - Prozessdefinition
 - Ausbildungsprogramm
 - Integriertes Softwaremanagement (Anpassung auf konkretes Projekt)
 - Software-Engineering-Techniken, Tool-Unterstützung
 - Koordination zwischen Gruppen
 - Peer Reviews

76 / 504

CMM Level 4 – Managed

- Einbeziehung quantitativer Aspekte
- Ziele setzen und überwachen
- Prozessmessdaten werden aufgenommen, archiviert, analysiert
- Vorhersagbarkeit steigt
- Key Process Areas:
 - Quantitative Prozesssteuerung
 - Leistung des Prozesses überwachen
 - Software-Qualitätsmanagement
 - Messbare Ziele für Prozessqualität

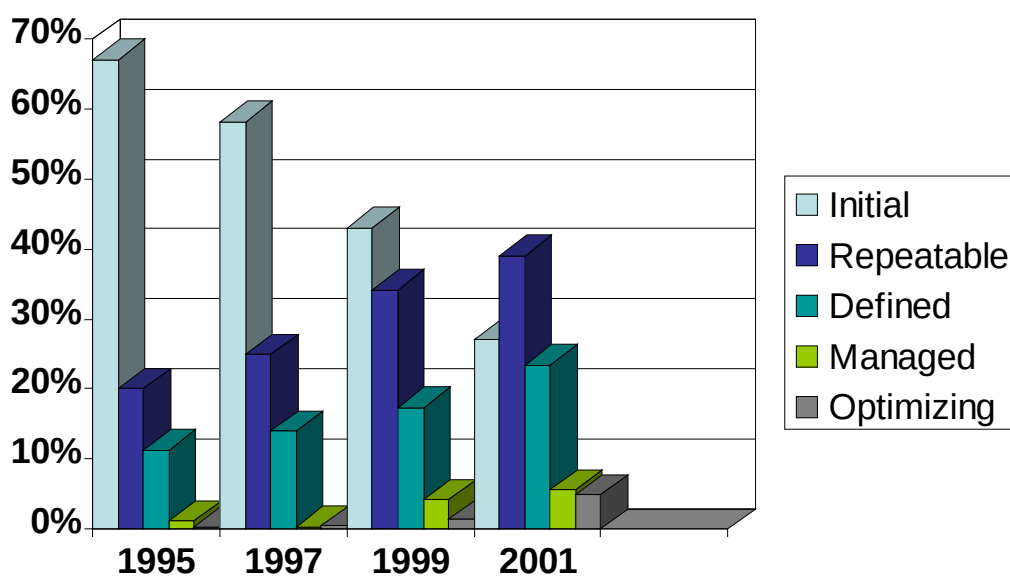
77 / 504

CMM Level 5 – Optimizing

- Änderungsmanagement für Technologie und Prozesse
- Feedback von Projekten zum Prozess → ständige Verbesserung
- Key Process Areas:
 - Defektvermeidung
 - Analyse von Fehler-/Problemursachen
 - Vermeiden erneuten Auftretens
 - Verwaltung von Technologieänderung
 - neue Technologien bewerten, evtl. einführen
 - Verwaltung von Prozessänderung
 - kontinuierliche Verbesserung

78 / 504

Capability Maturity Model



SEI Assessments (Quelle: SEI)

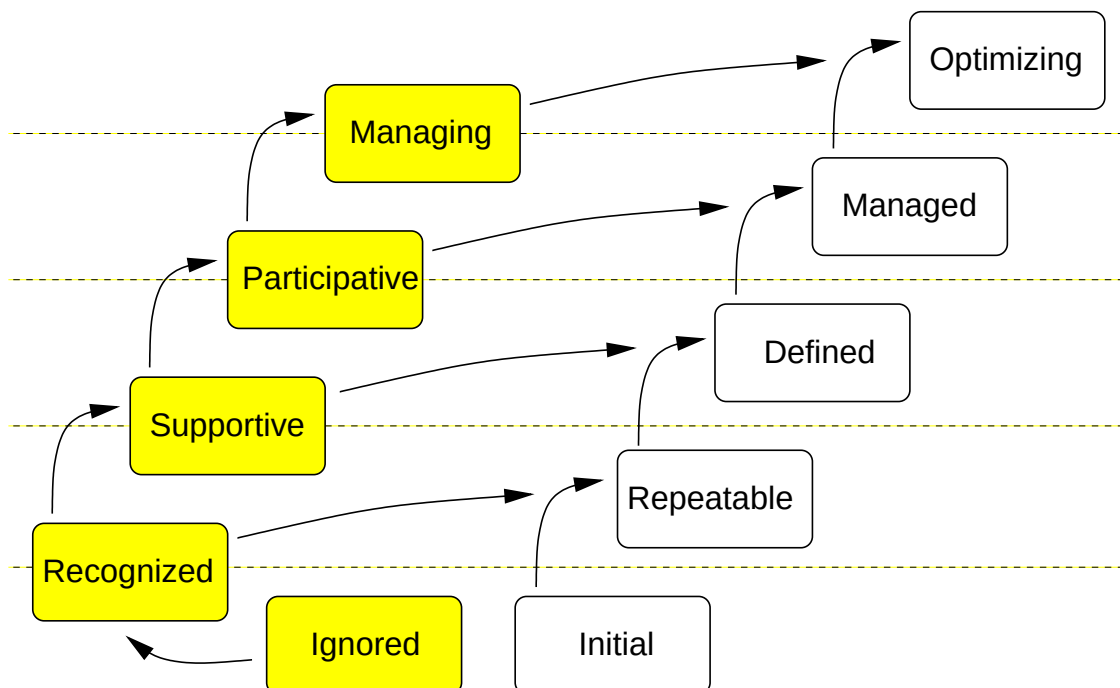
79 / 504

Probleme mit CMM

- Management: „zu teuer“
- Wenig verbreitet (ca. 10-15%)
- Nur langsame Verbesserung (ca. 2 Jahre/Level)
- Neue Technologien nicht berücksichtigt

80 / 504

Capability Maturity Model – Management



Quelle: Parker (1995)

81 / 504

Capability Maturity Model Integration (CMMI)

- Viele Maturity-Model-Varianten
- CMMI: „Integration“ (SEI 2000):
- Angepasst an iterative Entwicklung
- Generische Ziele hinzugefügt
- Zusätzliche KPAs:
 - Level 2: Measurement and Analysis
 - Level 3: Software Product Engineering (verfeinert); Risk Management; Decision Analysis and Resolution
 - Level 4, 5: nur Restrukturierungen

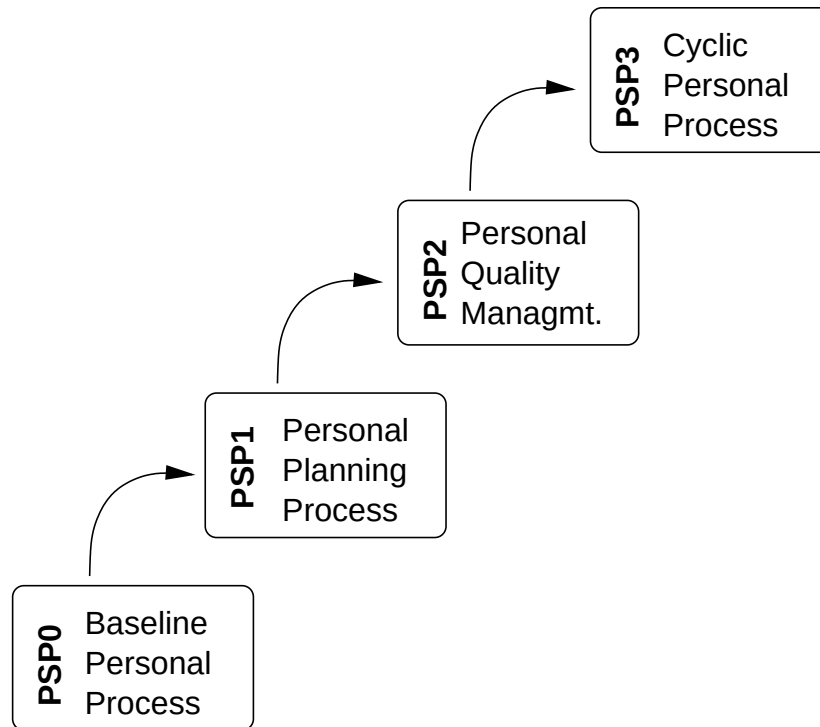
82 / 504

Persönlicher Softwareprozess (PSP)

- Watts S. Humphrey (1995) (SEI)
- Anwendung von CMM auf einen Entwickler
- Schwerpunkte: Planung, Qualität
- Vorteile:
 - Schneller umsetzbar
 - Konkrete Techniken angebbbar
 - Verbesserungen sofort wahrnehmbar
- höhere Motivation

83 / 504

PSP – Evolution



84 / 504

PSP0 – Baseline Personal Process

PSP0: Bisherige Vorgehensweise plus

- Messung
 - Zeit pro Phase
 - gefundene/gemachte Fehler pro Phase
 - Zeit für Fehlerbehebung
- Formulare: Projektplan, Zeiten, Fehler

PSP0.1: plus

- Codierrichtlinien
- Messung der LOC (Veränderungen)
- Formular: Prozessverbesserung

85 / 504

PSP1 – Personal Planning Process

PSP1: PSP0.1 plus

- Größenschätzung mit PROBE (PROxy-Based Estimating)
 - Schätzung basiert auf Objekten (Entwurf)
 - Unterscheidet Objekte nach Typ, Größe, #Methoden
 - Daten sammeln, Regressionsanalyse
- Formular: Testbericht

PSP1.1: PSP1 plus

- Aufgabenplanung
- Zeitschätzung, -planung
- Projektverfolgung (Earned Value Tracking)

86 / 504

PSP2 – Personal Quality Management

PSP2: PSP1.1 plus

- Code Reviews (Checklisten)
- Design Reviews (Checklisten)

PSP2.1: PSP2 plus

- Design Templates:
 - Operational Scenario ($\approx$ Anwendungsfall)
 - Functional Specification (formale Spezifikation)
 - State Specification ($\approx$ Zustandsdiagramm)
 - Logic Specification (Pseudocode)

→ Vermeidung von Designfehlern

→ Beurteilung der Qualität

- Cost-of-Quality (Behebung, Bewertung, Vermeidung)

87 / 504

PSP3:

- Anwendung auf große Projekte
- Nach High-Level Design: Aufteilung in Module
- Anwendung von PSP2.1 auf jedes Modul
- Formular: Issue tracking log

88 / 504

Wiederholungs- und Vertiefungsfragen

- Erläutern Sie die Ideen sowie Vor- und Nachteile der Entwicklungsprozesse. . .
 - Wasserfall
 - V-Modell
 - testgetriebene Entwicklung
 - inkrementelle Entwicklung
 - Spiralmodell
 - Rational Unified Process
 - Cleanroom Development
 - Extreme Programming
- Gegeben ist das folgende Szenario: [. . .]. Welches Vorgehensmodell empfehlen Sie?
- Unter welchen Umständen würden Sie eher agiles Vorgehen als voraus planendes empfehlen?
- Stellen Sie das Capability Maturity Model dar. Wozu dient es?
- Wie können Prozesse verbessert werden?
- Was ist der persönliche Software-Entwicklungsprozess? Wozu dient er?

89 / 504

Weiterführende Literatur

Bibliographie zu Entwicklungsprozessen:

Arbeitskreis der Fachgruppe 5.11

“Begriffe und Konzepte der Vorgehensmodellierung”

[http://www.vorgehensmodelle.de/giak/
arbeitskreise/vorgehensmodelle/publallg.html](http://www.vorgehensmodelle.de/giak/arbeitskreise/vorgehensmodelle/publallg.html)

Rational Unified Process: Kruchten (1998)

90 / 504

Software-Metriken

Software-Metriken

Messen und Maße

Skalen

Gütekriterien für Metriken

Vorgehensweise

Klassifikation von Softwaremetriken

Prozessmetriken

Ressourcenmetriken

Produktmetriken

Anwendungen

Probleme

Goal-Question-Metric-Ansatz

Wiederholungsfragen

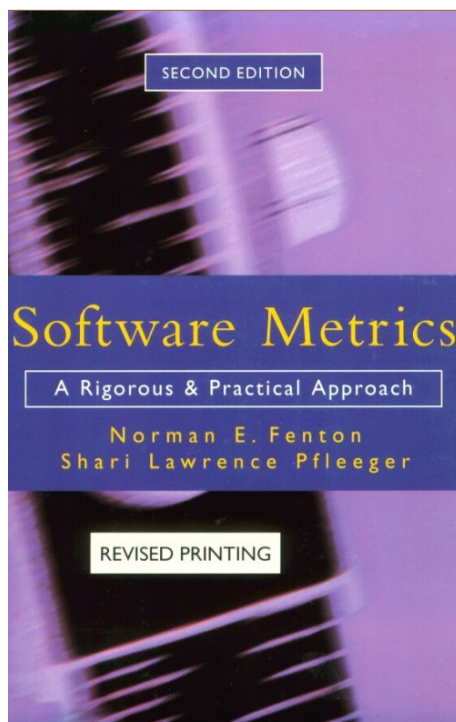
91 / 504

Lernziele

- Verstehen, was eine Software-Metrik ist
- die Einsatzmöglichkeiten von Metriken kennen
- Metriken beurteilen und auswählen können

92 / 504

Literatur



– Fenton und Pfleeger (1998)

93 / 504

Bedeutung des Messens

"To measure is to know."

Clerk Maxwell, 1831-1879

"A science is as mature as its measurement tools."

Louis Pasteur, 1822-1895

"'Miss alles, was sich messen lässt, und mach alles messbar, was sich nicht messen lässt.'"

Galileo Galilei, 1564-1642

"'Messen können Sie vieles, aber das Angemessene erkennen können Sie nicht.'"

Hans Gadamer

Messen spielt in allen Ingenieurwissenschaften eine wichtige Rolle.

Galilei: Ziel der Wissenschaft; durch Messung zu verständlicheren und nachprüfbaren Konzepten/Ergebnissen kommen.

Definitionen: Maß, Messen, Metrik

Definition

Maß:

Abbildung von einem beobachteten (empirischen)
Beziehungssystem auf ein numerisches Beziehungssystem

Abbildung von Eigenschaften von Objekten der realen Welt auf
Zahlen oder Symbole

Definition

Messen: Anwendung eines Maßes auf ein Objekt

Definition

Metrik: Abstandsmaß (math.)

95 / 504

Definitionen für Software-Metriken

*“A quantitative measure of the degree to which a system,
component, or process possesses a given variable.”*

– IEEE Standard Glossary

*“A software metric is any type of measurement which relates
to a software system, process or related documentation.”*

– Ian Sommerville

96 / 504

Messen und Softwaretechnik

- Beschreibung: kompakt und objektiv
- Beurteilung: Vergleich, Verbesserungen
- Vorhersage: geordnete Planung, Verbesserungen

97 / 504

Fragen bei Maßen

- | | |
|-----------------|--|
| • Repräsentanz | Darstellung als Zahl sinnvoll möglich? |
| • Eindeutigkeit | viele Abbildungen möglich |
| • Bedeutung | erhalten bei Transformationen |
| • Skalierung | welche Skala? |

98 / 504

Worüber wir uns bei der Definition von Metriken Gedanken machen müssen.

There are three important questions concerning representations and scales:

1. How do we determine when one numerical relation system is preferable to another?
2. How do we know if a particular empirical relation system has a representation in a given numerical relation system?
3. What do we do when we have several different possible representations (and hence many scales) in the same numerical relation system?

Question 2 is known as the representation problem.

Skalen

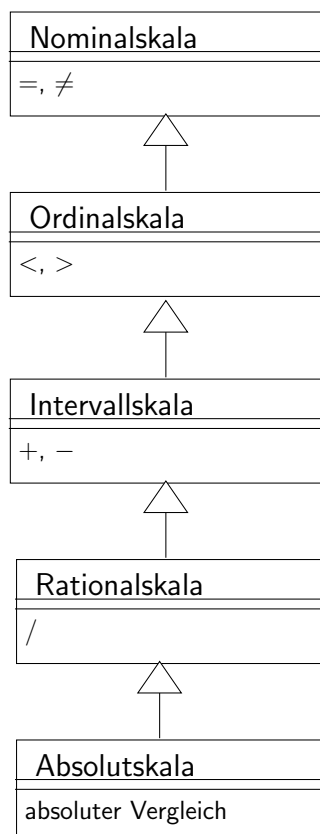
- ① „20 Prozent Verbesserung der Qualität“
- ② „Dieser Kunde ist doppelt so zufrieden wie jener“
- ③ „Heute doppelt so warm wie gestern“
(Temperatur gestern: 10°C; heute: 20°C)

- ① Was ist Qualität Null?
- ② Wie zufrieden sind sie denn?
- ③ $10^{\circ}\text{C} \rightarrow 20^{\circ}\text{C} \hat{=} +3,5\%$
denn $10^{\circ}\text{C} = 283 \text{ Kelvin}$, $20^{\circ}\text{C} = 293 \text{ Kelvin}$

→ Skala?

99 / 504

Skalenhierarchie



100 / 504

Skalenhierarchie – Nominalskala

1. Nominalskala

- ungeordnete 1:1 Abbildung
- Transformationen: beliebige 1:1
- Operationen: $=$, $\neq$
- Statistiken: Häufigkeit
- Beispiel: Programmiersprachen

Ada C C++ Java ...

101 / 504

Skalenhierarchie – Ordinalskala

2. Ordinalskala

- dazu: vollständige Ordnung
- Transformationen: streng monoton steigend
- Operationen: $<$, $>$
- Statistiken: Median
- Beispiel: Prioritäten

niedrig $<$ mittel $<$ hoch

102 / 504

3. Intervallskala

- dazu: Distanzfunktion
- Transformationen: $M' = aM + b$ ($a > 0$)
- Operationen: $+$, $-$
- Statistiken: Mittelwert, Standardabweichung
- Beispiel: Temperatur

$$T_{Celsius} = \frac{5}{9} \cdot (T_{Fahrenheit} - 32)$$

103 / 504

Definition Metrik

Metrik: Distanzfunktion $d : A \times A \rightarrow \mathbb{R}$, mit:

- $d(a, b) \geq 0 \quad \forall a, b \in A, \quad d(a, b) = 0 \Leftrightarrow a = b$
- $d(a, b) = d(b, a) \quad \forall a, b \in A$
- $d(a, c) \leq d(a, b) + d(b, c) \quad \forall a, b, c \in A$

104 / 504

4. Rationalskala

- dazu: Maßeinheit, Nullpunkt
- Transformationen: $M' = aM$ ($a > 0$)
- Operationen: /
- Statistiken: geom. Mittel, Korrelation
- Beispiel: Länge

$$L_{Meter} = L_{Meilen} \cdot 1609$$

Das geometrische Mittel zwischen zwei Zahlenwerten ist: $\sqrt{f_1 \cdot f_2}$
Das arithmetische Mittel zwischen zwei Zahlwerten ist: $(f_1 + f_2)/2$

Skalenhierarchie – Absolutskala

5. Absolutskala

- Metrik steht für sich selbst, kann nicht anders ausgedrückt werden
- Transformationen: nur die Identität $M' = M$
- Operationen: absoluter Vergleich; d.h.
 - es existiert ein natürlicher Nullpunkt
 - und Maßeinheit ist natürlich gegeben (d.h. im weitesten Sinne 'Stück')
- Beispiele:
 - Zähler: Anzahl Personen in einem Projekt
 - Wahrscheinlichkeit eines Fehlers
 - LOC für Anzahl Codezeilen
 - **nicht:** LOC für Programmlänge

106 / 504

Gütekriterien für Metriken

- Objektivität
- Validität
- Zuverlässigkeit
- Nützlichkeit
- Normiertheit
- Vergleichbarkeit
- Ökonomie

– Balzert (1997)

107 / 504

(Güte entspr. Qualität)
Objekt.: kein subjektiver Einfluss durch Messenden möglich
Valid.: misst wirklich das, was sie vorgibt zu messen
Zuverläss.: Wiederholung liefert gleiche Ergebnisse
Nützl.: hat praktische Bedeutung
Norm.: es gibt eine Skala für die Messergebnisse
Vergl.: mit anderen Maßen vergleichbar
Ökon.: mit vertretbaren Kosten messbar

Vorgehensweise

- ① Definition
 - Zielbestimmung
 - Modellbildung
 - Skalentypbestimmung
 - Maßdefinition
- ② Validierung
- ③ Anwendung
 - Konkretes Modell bilden
 - Messung
 - Interpretation
 - Schlussfolgerung

Validierung von Maßen

Interne Validierung:

Nachweis, dass ein Maß eine gültige numerische Charakterisierung des entsprechenden Attributs ist, durch

- Nachweis der Erfüllung der Repräsentanzbedingung
- und Prüfung des Skalentyps

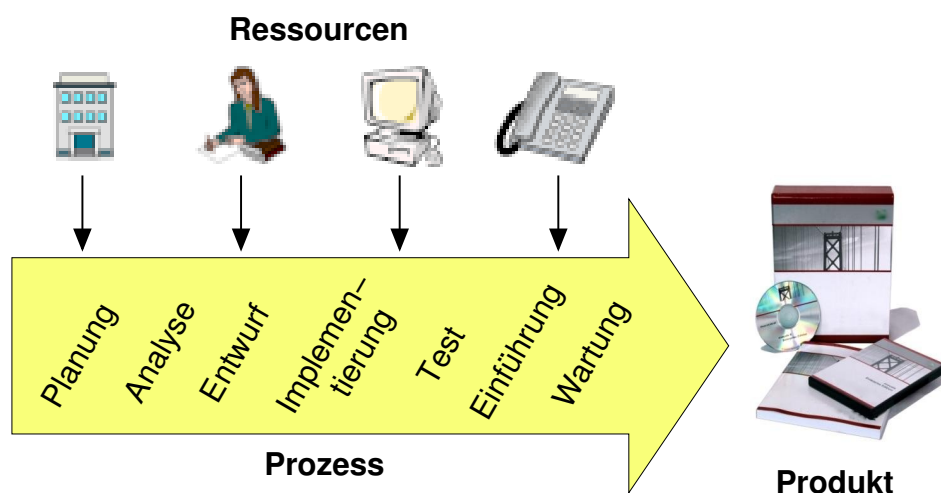
Externe Validierung → Vorhersagemodell:

- Hypothese über Zusammenhang zwischen zwei Maßen
- Erfassung der Meßwerte beider Maße auf gleicher Testmenge
- Statistische Analyse der Ergebnisse
 - Bestimmung von Parametern
 - Prüfung der Allgemeingültigkeit

109 / 504

Klassifikation von Softwaremetriken

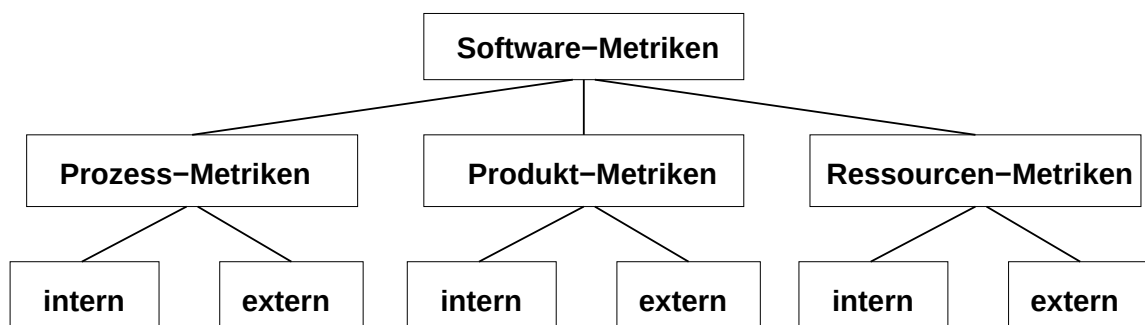
- Was: Ressource/Prozess/Produkt
- Wo: intern/extern (isoliert/mit Umgebung)
- Wann: in welcher Phase des Prozesses
- Wie: objektiv/subjektiv, direkt/abgeleitet



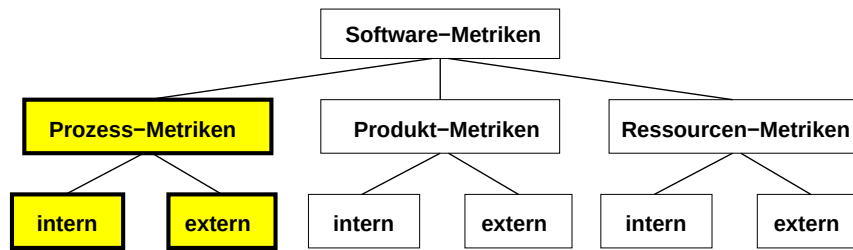
110 / 504

Bei den Metriken unterscheidet man zwischen internen und externen Metriken. Eine interne Metrik ist darüber definiert, dass sie nur Eigenschaften innerhalb des untersuchten Objektes misst, wohingegen externe Metriken die Interaktion des Objektes mit seiner Umgebung berücksichtigen.

Klassifikation nach Fenton und Pfleeger (1998)



Prozessmetriken



intern:

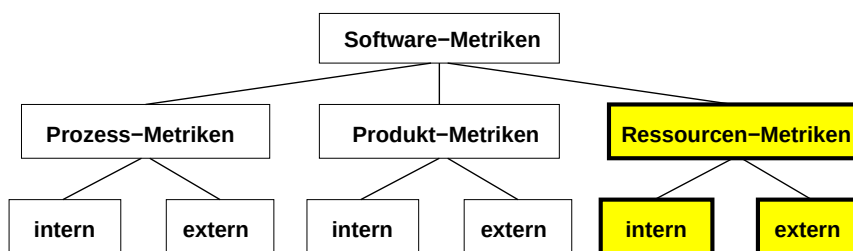
- Zeit/Dauer
- Aufwand
- Anzahl von Ereignissen
z.B. Fehler, Änderungen

extern:

- Qualität
- Kontrollierbarkeit
- Stabilität
- Kosten

112 / 504

Ressourcenmetriken



intern:

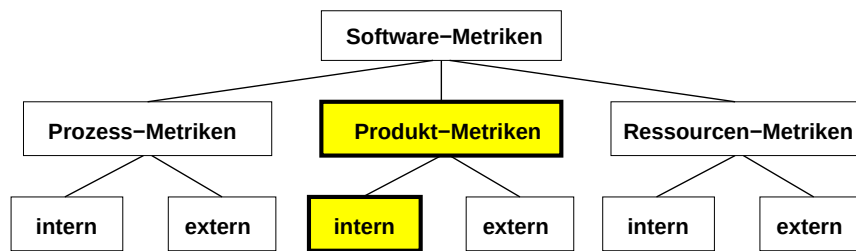
- Personal (Alter, Lohn)
- Teamgröße/-struktur
- Produktionsmaterialien
- Werkzeuge, Methoden

extern:

- Produktivität
- Erfahrung
- Kommunikation
- ...

113 / 504

Produktmetriken – intern



Größe:

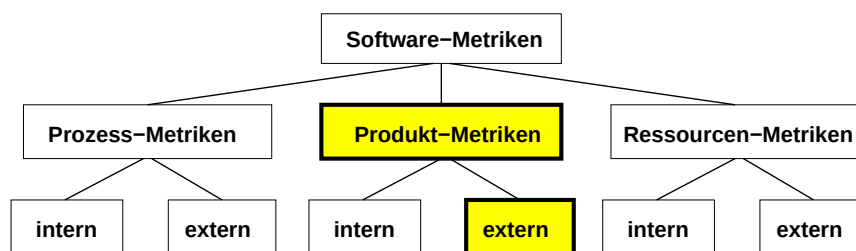
- LOC
- Halstead
- Function Points
- Bang (DeMarco)

Komplexität:

- McCabe Cyclomatic Complexity
- Kontrollflussgraph
- Datenfluss
- OO-Metriken

114 / 504

Produktmetriken – extern



- Zuverlässigkeit
- Verständlichkeit
- Benutzerfreundlichkeit
- Performanz
- Portierbarkeit
- Wartbarkeit
- Testbarkeit
- ...

115 / 504

Produktmetriken – intern

Vorteil: automatische Erfassung

Die Klassiker:

- LOC - Lines Of Code
- Halstead (1977)
- McCabe (1976)
- OO-Metriken (Chidamber und Kemerer 1994)

116 / 504

Größenmetriken – LOC

Lines of code (LOC)

- + relativ einfach messbar
- + starke Korrelation mit anderen Maßen
- ignoriert Komplexität von Anweisungen und Strukturen
- schlecht vergleichbar
- abgeleitet: Kommentaranteil

117 / 504

Physical source lines (COCOMO 2.0)

When a line or statement contains more than one type, classify it as the type with the highest precedence.

Statement type	Precedence	Included
Executable	1	✓
Nonexecutable		
Declarations	2	✓
Compiler directives	3	
Comments		
On their own lines	4	
On lines with source code	5	
Banners and non-blank spacers	6	
Blank (empty) comments	7	
Blank lines	8	

118 / 504

Physical source lines (COCOMO 2.0)

How produced	Included
Programmed	✓
Generated with source code generators	
Converted with automated translators	✓
Copied or reused without change	✓
Modified	✓
Removed	

119 / 504

Physical source lines (COCOMO 2.0)

Origin	Included
New work: no prior existence	✓
Prior work: taken or adapted from	✓
A previous version, build, or release	✓
Commercial off-the-shelf software (COTS), other than libraries	
Government furnished software (GFS), other than reuse libraries	
Another product	
A vendor-supplied language support library (unmodified)	
A vendor-supplied operating system or utility (unmodified)	
A local or modified language support library or operating system	
Other commercial library	
A reuse library (software designed for reuse)	✓
Other software component or library	✓

120 / 504

Anwendungen

- Beurteilung des aktuellen Zustands
 - Projektüberwachung
 - Produktivität
 - Softwarequalität
 - Prozessqualität (CMM)
- Vorhersage des zukünftigen Zustands
 - Aufwandsabschätzung
 - Prognose für Wartungskosten

121 / 504

Probleme

- Datenerfassung sehr aufwendig, zunächst wenig Nutzen
- Datenerfassung nicht konsistent
- Teilweise Messungen schwierig durchführbar
- Zweck der Messungen muss klar sein
- Integration der Datenerfassung in den normalen Arbeitsprozess
- Metriken müssen wohldefiniert und validiert sein
- Beziehungen zwischen Metriken müssen definiert sein
- Gefahr der Fehlinterpretation

122 / 504

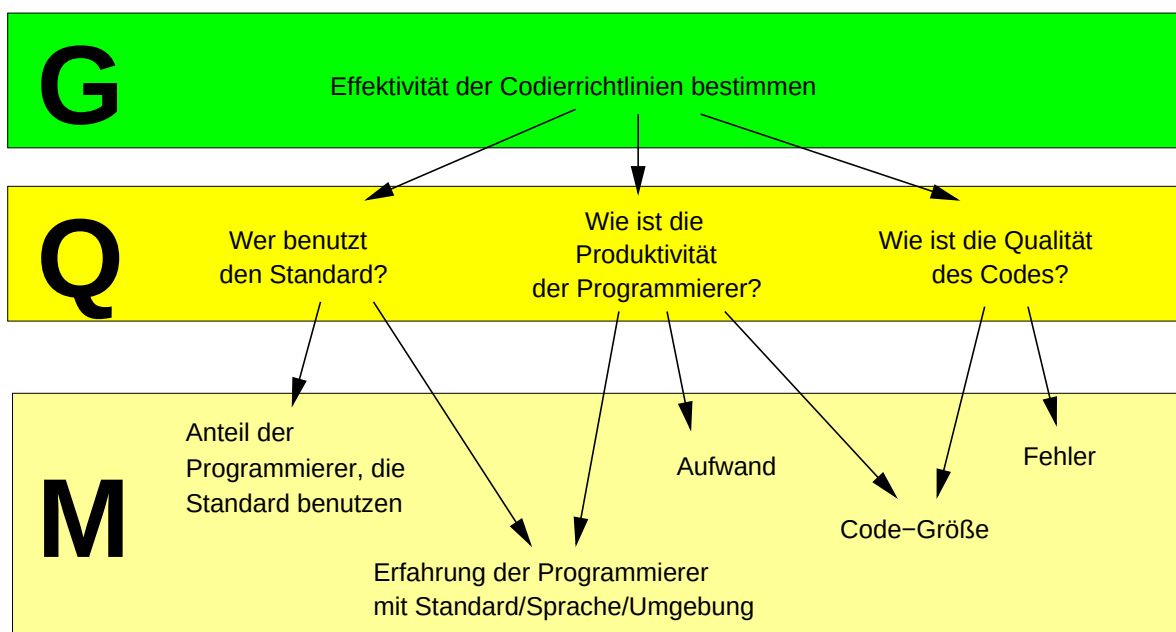
Zielorientiertes Messen

Goal-Question-Metric (GQM) (Basili und Weiss 1984))

- ① Ziele erfassen
- ② zur Prüfung der Zielerreichung notwendige Fragen ableiten
- ③ was muss gemessen werden, um diese Fragen zu beantworten

123 / 504

Zielorientiertes Messen



Beispiel: Prozess

Ziel	Frage	Metrik
Maximiere Kundenzufriedenheit	Wie viele Probleme treten beim Kunden auf?	• #Fehler (FR) und #Änderungswünsche (ÄR) • Zuverlässigkeit • Break/Fix-Verhältnis
	Wie lange dauert Problembehebung?	• Verhältnis und Dauer offener und geschlossener FR/ÄR
	Wo sind Flaschenhälse?	• Personaleinsatz • Nutzung anderer Ressourcen

125 / 504

Wiederholungs- und Vertiefungsfragen

- Was ist ein Maß? Was ist eine Metrik?
- Was ist eine Software-Metrik?
- Welche Skalen gibt es für Daten? Welche Eigenschaften haben diese?
- Beschreiben Sie das Vorgehen bei der Definition und Einführung eines Maßes. Was unterscheidet die interne von der externen Validierung?
- Wie lassen sich Software-Metriken klassifizieren? Nennen Sie Beispiele für jede Klasse.
- Was ist die Bedeutung von Metriken im Software-Entwicklungsprozess?
- Was ist die GQM-Methode? Erläutern Sie GQM anhand des Zieles X.

N.B.: Die Übungsaufgaben sind weitere Beispiele relevanter Fragen.

126 / 504

Empirische Softwaretechnik

Empirische Softwaretechnik

Motivation

Wissenserwerb in Wissenschaft und Engineering

Untersuchungsmethoden

Bestandteile eines Experiments

127 / 504

Lernziele

- die Notwendigkeit zur empirischen Forschung in der Softwaretechnik erkennen
- prinzipielles Vorgehen verstehen
- (irgendwann einmal) empirisch forschen können

128 / 504

Experimentation in software engineering is necessary but difficult. Common wisdom, intuition, speculation, and proofs of concept are not reliable sources of credible knowledge.

– V.R. Basili, 1999

129 / 504

Motivation

- Wir wollen genau wissen, ob und unter welchen Randbedingungen eine Methode funktioniert.
- Forschung
 - beweist durch logische Schlüsse
 - oder aber beobachtet, experimentiert und misst.
- Messung ist sorgfältige Beobachtung mit größtmöglicher Präzision, Zuverlässigkeit und Objektivität
- Messungen identifizieren neue Phänomene, testen Hypothesen oder leiten uns bei der Anwendung von Modellen und Methoden

Empirische Untersuchungen

Methoden, die von Menschen angewandt werden, können nur empirisch untersucht werden.

130 / 504

Experiment von Knight und Leveson (1986):

Hypothese: N-Version-Programming führt zu zuverlässigerer Software.

Experiment von Dzidek u. a. (2008):

Hypothese: Dokumentation in UML hilft bei der Wartung.

Knight and Leveson's experiment analyzed the failure probabilities of multiversion programs. Conventional theory predicted that the failure probability of a multiversion program was the product of the failure probabilities of the individual versions. However, John Knight and Nancy Leveson observed that real multiversion programs had significantly higher failure probabilities. In essence, the experiment falsified the basic assumption of the conventional theory, namely that faults in program versions are statistically independent.

– Tichy (1998) This is the first controlled experiment that investigates the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real nontrivial system, using professional developers as subjects, working with a state-of-the-art UML tool during an extended period of time. The subjects in the control group had no UML documentation. In this experiment, the subjects in the UML group had, on average, a practically and statistically significant 54 percent increase in the functional correctness of changes ($p=0.03$) and an insignificant 7 percent overall improvement in design quality ($p=0.22$), though a much larger improvement was observed on the first change task (56 percent), at the expense of an insignificant 14 percent increase in development time caused by the overhead of updating the UML documentation ($p=0.35$).

– Dzidek u. a. (2008)

Beispiel von Müller (2006) I

Objective:

Comparison of program defects caused by programmer pairs and solo developers.

Design:

Analysis of programs developed during two counter balanced experiments.

Setting:

Programming lab at University.

Experimental Units:

42 programs developed by computer science students participating in an extreme programming lab course.

Main Outcome Measures:

Programmer pairs make as many algorithmic mistakes but fewer expression mistakes than solo programmers.

132 / 504

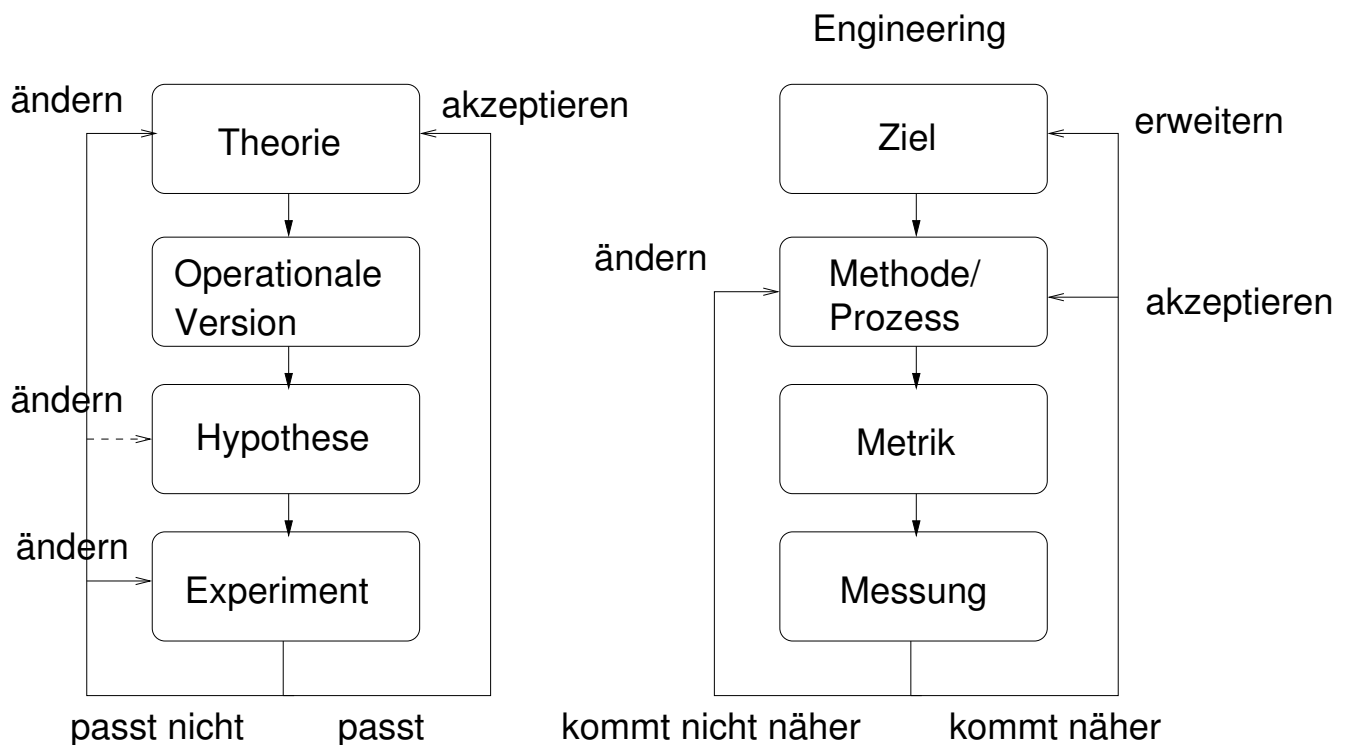
Beispiel von Müller (2006) II

Results: The second result is significant on the 5 percent level.

Conclusions: For simple problems, pair programming seems to lead to fewer mistakes than solo programming.

133 / 504

Wissenserwerb in Wissenschaft und Engineering



134 / 504

Beschaffenheit empirischer Forschung

- in der Softwaretechnik erst seit ungefähr 1980 als wesentliche Disziplin anerkannt
- heute: Konferenzen und Zeitschriften
- Verschiebung weg von rein mathematischen Methoden
- Mehrzahl der Probleme der Softwaretechnik sind nicht mathematischer Art, hängen vielmehr von Menschen ab

136 / 504

Untersuchungsmethoden

- Umfragen und Erhebungen
- Geschichte
- Archivarische Analyse
- Fallstudien
- kontrollierte Experimente

137 / 504

Untersuchungsmethoden

Umfragen und Erhebungen:

- Datenerfassung mit Fragebögen oder ethnographische Studien
- können auch a-posteriori durchgeführt werden
- dienen oft der Bildung von Hypothesen für nachfolgende Fallstudien oder kontrollierte Experimente
- fundierte Methoden zur Datenerhebung und -validierung notwendig
- entstammen den Sozialwissenschaften und kognitiven Wissenschaften

138 / 504

Untersuchungsmethoden

Definition einer Fallstudie nach Yin (2003):

A case study is an empirical inquiry that

- *investigates a contemporary phenomenon within its real-life context, especially when*
- *the boundaries between phenomenon and context are not clearly evident*

The case study inquiry

- *cope with the technically distinctive situation in which there will be many more variables of interest than data points, and as one result*
- *relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result*
- *benefits from the prior development of theoretical propositions to guide data collection and analysis.*

139 / 504

Untersuchungsmethoden

Fallstudien (Quasi-Experimente):

- In-Vivo-Experiment oder Feldstudien mit Hypothese
- eingebettet in echte Projekte und damit weniger kontrolliert
- Herausforderung: zu messen, ohne den Verlauf zu verfälschen

140 / 504

kontrollierte Experimente (In-Vitro-Experiment):

- finden in kontrollierter Testumgebung statt
- Hypothese wird falsifiziert oder bestätigt (mit einer gewissen Wahrscheinlichkeit)
- unabhängige Variablen: Eingabeparameter, die im Experiment variiert werden
- abhängige Variablen: Ausgabeparameter, die gemessen werden

141 / 504

Übersicht über Untersuchungsmethoden

	Art der Frage	mögliche Kontrolle	Fokus auf Gegenwart
Experimente	wie, warum?	ja	ja
Umfragen und Erhebungen	wer, was, wo, wie viele, wie sehr?	nein	ja
Archivarische Analyse	wer, was, wo, wie viele, wie sehr?	nein	ja/nein
Geschichte	wie, warum?	nein	nein
Fallstudien	wie, warum?	nein	ja

Quelle: COSMOS Corporation, erläutert von Yin (2003)

142 / 504

Bestandteile eines Experiments

1. Festlegung der Ziele

Aspekte:

- ① Objekt der Studie (z.B. Entwurf, Codierung, Test)
- ② Zweck der Studie (z.B. Vergleich, Analyse, Voraussage)
- ③ Fokus der Studie (z.B. Effektivität, Effizienz)
- ④ Standpunkt (z.B. Praktiker, Forscher)
- ⑤ Kontext (z.B. Erfahrung der Teilnehmer, verwendete Elemente, Umgebung)

Kontext → unabhängige Variablen

Fokus → abhängige Variablen

144 / 504

Bestandteile eines Experiments

2. Formulierung einer testbaren Hypothese

„Methode M ist geeignet für Aufgabe A“

versus

„Methode M benötigt weniger Zeit als Methode N, um Aufgabe A zu erledigen“

Null-Hypothese (Negation der Hypothese):

„Methode M benötigt die gleiche oder mehr Zeit als N, um Aufgabe A zu erledigen“

Wenn Null-Hypothese widerlegt ist, wird Hypothese bestätigt (mit einem gewissen Grad an Konfidenz)

145 / 504

Bestandteile eines Experiments

3. Aufbau des Experiments

- Korrelation versus Kausalität
- Validität
 - interne: es werden tatsächlich nur die Beziehungen zwischen unabhängigen und abhängigen Variablen gemessen (z.B. Lerneffekte, zeitliche Aspekte, Auswahl der Teilnehmer)
 - externe: Übertragbarkeit (z.B. Studenten versus Praktiker)
- Identifikation aller unabhängiger und abhängiger Variablen
- Randomisierung

→ viele Bücher zu Standard-Designs abhängig von den Rahmenbedingungen

146 / 504

Bestandteile eines Experiments

4. Analyse und Validierung der Resultate

- Auswertung durch statistische Methoden (Korrelationen und Regressionsanalyse)
- Problem des geringen Datenumfangs
 - parametrische statistische Tests setzen bestimmte Verteilung voraus
 - meist nur nicht-parametrische statistische Tests adäquat, weil keine Verteilung voraus gesetzt wird (insbesondere für Nominal- bis Ordinalskalen)
- quantitative Analyse oft ergänzt durch qualitative
- Validierung
 - Befragungen der Teilnehmer, um sicher zu stellen, dass sie alle Fragen verstanden haben
 - Untersuchung statistischer Ausreißer
- Benchmarking schwierig, weil Firmen ihre Daten ungern veröffentlichen

147 / 504

Bestandteile eines Experiments

5. Replikation

- Wiederholung, um „Glückstreffer“ auszuschließen
- Experiment muss detailliert beschrieben sein
- bedauerlicherweise schwer zu veröffentlichen, weil keine neuen Ergebnisse präsentiert werden

148 / 504

Ethische Fragen

- keine Teilnahme ohne explizite Einwilligung
- kein Missbrauch
- Anonymität muss gewährt werden (doppelt blind)
- kein materieller oder sonstiger Gewinn (Bezahlung, Gehaltserhöhung, gute Note etc.)

149 / 504

Grenzen der experimentellen Forschung

- hoher Aufwand (allerdings: Lernen ohne Experimente ist auch teuer)
- Abhängigkeit von menschlichen Versuchsteilnehmern
 - Studenten haben möglicherweise nicht die Erfahrung
 - Praktiker haben wenig Zeit
- Transferierbarkeit der Resultate
 - Software-Projekte haben eine Unzahl möglicher Variablen
 - nicht alle sind kontrollierbar
 - und wenn sie kontrolliert sind, treffen sie möglicherweise nicht auf andere Umgebungen zu

150 / 504

Kontrollierte Experimente

Kontrollierte Experimente

Begriffe

Auswahl der Teilnehmer

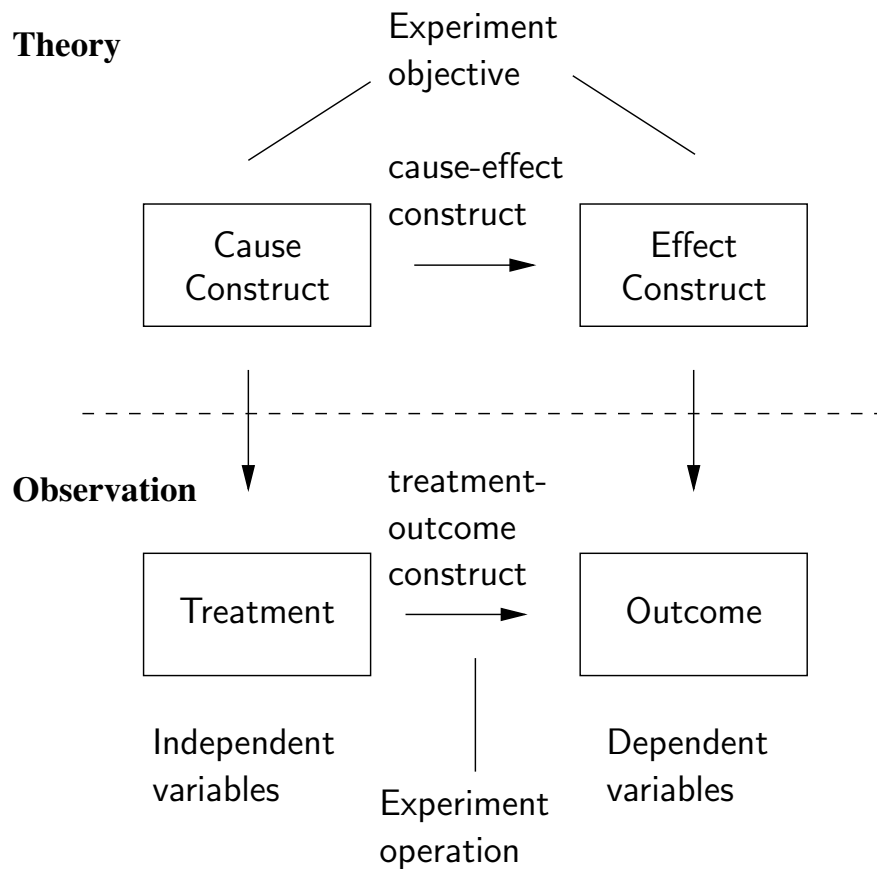
Aufbau von Experimenten

Gültigkeit (Threats to Validity)

Wiederholungsfragen

151 / 504

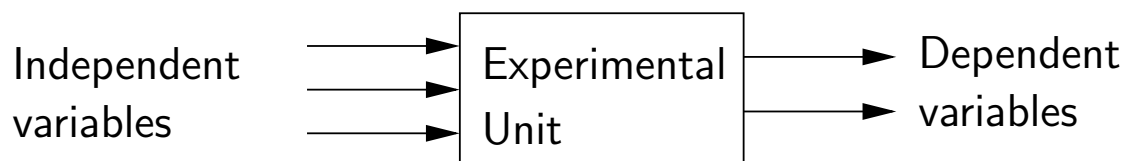
Theorie und Beobachtung (Wohlin u. a. 2000)



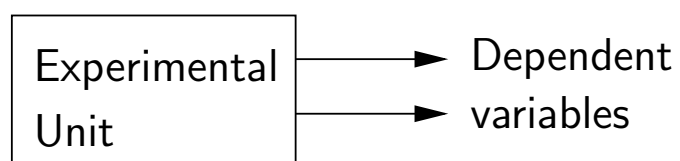
152 / 504

Experiment

Experiment



Observational Study



153 / 504

Definition

Experiment: An investigation in which the investigator applies some treatments to experimental units and then proceeds to observe the effect of the treatments on the experimental units by measuring one or more dependent (response) variables.

Definition

Observational Study: an investigation in which the investigator observes units and measures one or more response variables without imposing treatments on the individuals.

Zieldefinition eines Experiments

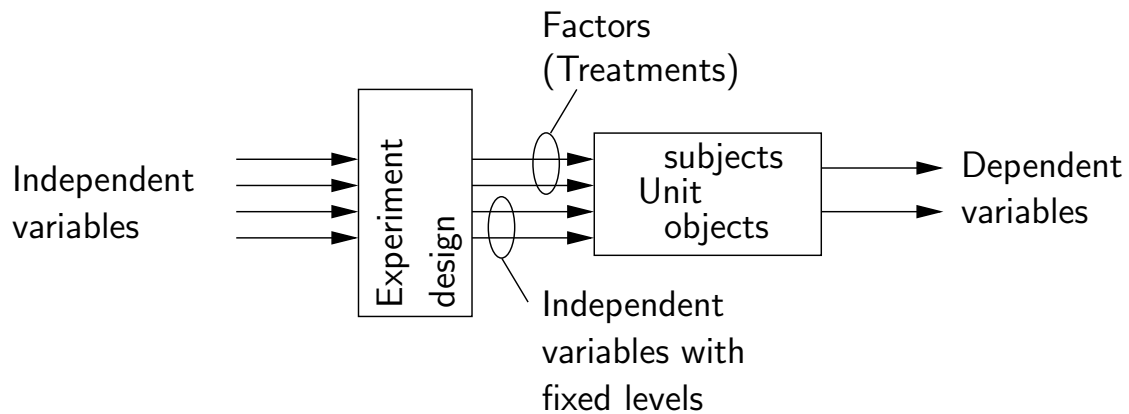
Vorlage:

Analyse *Object of Study*
for the purpose of *Purpose*
with respect to *Quality focus*
from the point of view of the *Perspective*
in the context of *Context*.

Beispiel:

Analyse *Quality assurance process*
for the purpose of *Characterize code inspection onto quality*
with respect to *Effectiveness and Cost*
from the point of view of the *Researcher*
in the context of *Professional developers in a software organization*.

Experiment



155 / 504

Definition

Independent Variable: variable that can be controlled and changed in the experiment.

z.B. Code-Inspektion, Erfahrung der Gutachter, inspizierte Software.

Definition

Factor: an explanatory variable studied in an investigation that can be set at any one of two or more values.

z.B. Code-Inspektion

Definition

Levels: the different values of a factor.

z.B. Code-Inspektion: angewandt oder nicht

Definition

Treatment: the circumstances created for an experiment, i.e., one particular set of values of factors.

z.B. Code-Inspektion wird angewandt

Definition

Object: the entity that receives the treatment.

z.B. die Software

Definition

Subject (Participant): the person that applies the treatment.

z.B. Entwickler

Definition

Test (Trial): a combination of treatment, subject, and object. An experiment consists of a set of tests.

z.B. Entwickler wendet Code-Inspektion für Software an.

Definition

Dependent Variable (Response Variable): a characteristic of an experimental unit measured after treatment and analyzed to address the objectives of the experiment.

z.B. Anzahl gefundener Fehler und Dauer

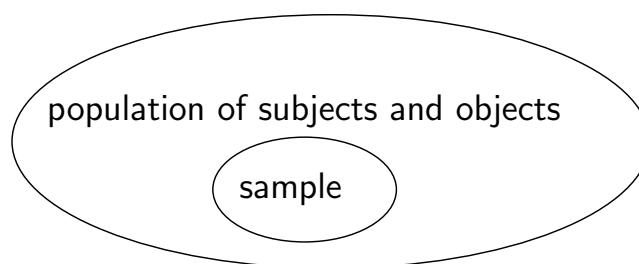
Definition

Experimental Error: accounts for the fact that experimental units treated independently and identically will not have identical dependent variable measurements.

z.B. Tagesform der Entwickler führt zu unterschiedlichen Messungen.

Auswahl der Teilnehmer

Sampling: Auswahl/Stichprobe von Objects und Subjects



Aspekte:

- Auswahlgröße beeinflusst experimentellen Fehler und Aussagekraft des statistischen Tests
 - je höher die Varianz in der Population, desto größer muss die Auswahl sein
 - Art der späteren Analyse beeinflusst Auswahlgröße
- Art der Analyse frühzeitig festlegen

Auswahl der Teilnehmer

Sampling-Strategien:

- Quasi-Experiment: Subjects nicht zufällig ausgewählt
- Convenience Sampling: Auswahl nach Einfachheit
- Simple Random Sampling: zufällige Auswahl
- Systematic Sampling: Population wird angeordnet; erstes Subject zufällig gewählt, dann jedes n'te Subject

→ Annahme: Population der Subjects ist homogen

157 / 504

Auswahl der Teilnehmer

Partitionierende Sampling-Strategien:

Partitionierung der Population in homogene Gruppen

- Quota Sampling: feste Quota für die Auswahl aus Partitionen ist vorgegeben, dann Convenience Sampling für einzelne Gruppen
- Stratified Random Sampling
 - Proportionate Stratified Random Sampling: zufällige Auswahl aus jeder Gruppe ist proportional zur Gesamtpopulation; Bsp.: 60 % Männer, 40 % Frauen → 3 Männer, 2 Frauen
 - Optimum (Disproportionate) Stratified Random Sampling: zufällige Auswahl aus Gruppe ist proportional zur Standardabweichung der Verteilung der Variable (mehr Auswahlen für die Gruppen mit höherer Verschiedenheit)

158 / 504

Generelle Prinzipien

- Randomization: Beobachtungen betreffen unabhängige Zufallsvariablen (Zuordnung Treatments–Subjects–Objects und Reihenfolge der Treatments)
- Blocking:
 - ① Ähnliche Objekte werden gruppiert
 - ② Treatments werden durch Vergleich der abhängigen Variablen desselben Blocks evaluiert
- Balancing: jedes Treatment hat gleiche Anzahl von Subjects
- Replication: mindestens ein Treatment wird unabhängig auf zwei oder mehr Objekte angewandt

159 / 504

Experimental Design

Arten von Studien:

		# objects	
		1	>1
# subjects per object	1	single-object	multi-object variation
	>1	multi-test within object	blocked subject-object

160 / 504

Definition

Full Factorial Treatment Design: the treatments consist of all possible combinations of the levels of the factors of interest.

- ein Faktor mit zwei Treatments
- ein Faktor mit mehr als zwei Treatments
- zwei Faktoren mit zwei Treatments
- mehr als zwei Faktoren mit mehr als zwei Treatments

161 / 504

Ein Faktor mit zwei Treatments

Definition

Completely randomized design: alle möglichen Zuordnungen von Treatments und Subjects/Objects sind gleich wahrscheinlich.

Subjects	Treatment 1	Treatment 2
1	X	
2		X
3		X
4	X	
5	X	
6		X

μ_i = Mittelwert der abhängigen Variablen für Treatment i

Nullhypothese: $\mu_1 = \mu_2$

Statistische Tests: t-test, Mann-Whitney

162 / 504

Ein Faktor mit zwei Treatments

Definition

Paired comparison design: jedes Subject wendet alle Treatments an. Reihenfolge wird randomisiert.

Subjects	Treatment 1	Treatment 2
1	1	2
2	2	1
3	2	1
4	1	2
5	1	2
6	2	1

y_{ij} = j'te Messung der abhängigen Variablen für Treatment i

$d_j = y_{1j} - y_{2j}$ und μ_d = Mittelwert von d

Nullhypothese: $\mu_d = 0$

Statistische Tests: Paired t-test, Sign test, Wilcoxon

163 / 504

Ein Faktor mit mehr als zwei Treatments

Completely randomized design

Subjects	Treatment 1	Treatment 2	Treatment 3
1	X		
2		X	
3			X
4	X		
5			X
6		X	

Nullhypothese: $\mu_1 = \mu_2 = \dots = \mu_n$

Statistische Tests: ANOVA, Kruskal-Wallis

164 / 504

Ein Faktor mit mehr als zwei Treatments

Paired comparison design

Subjects	Treatment 1	Treatment 2	Treatment 3
1	1	2	3
2	1	3	2
3	2	1	3
4	2	3	1
5	3	1	2
6	3	2	1

Nullhypothese: $\mu_1 = \mu_2 = \dots = \mu_n$

Statistische Tests: ANOVA, Kruskal-Wallis

165 / 504

Zwei Faktoren

Definition

2 × 2 Factorial Design: zufällige Zuordnung der Subjects für jede Kombination der Treatments

		Faktor A	
		Treatment A1	Treatment A2
Faktor B	Treatment B1	4,6	1,7
	Treatment B2	2,3	5,8

α_i = Effekt von Treatment i auf Faktor A

β_i = Effekt von Treatment i auf Faktor B

$(\alpha\beta)_{ij}$ = Effekt der Interaktion von α_i und β_j

Nullhypothesen: $\alpha_1 = \alpha_2$ oder $\beta_1 = \beta_2$ oder $(\alpha\beta)_{ij} = 0$ für alle i, j

Statistische Tests: ANOVA

166 / 504

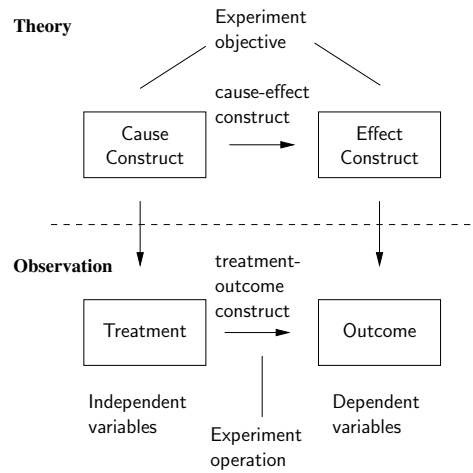
Mehr als zwei Faktoren mit zwei Treatments

Definition

2^k Factorial Design für k Faktoren: zufällige Zuordnung der Subjects für jede der 2^k Kombinationen der Treatments

Faktor A	Faktor B	Faktor C	Subjects
A1	B1	C1	2,3
A2	B1	C1	1,13
A1	B2	C1	5,6
A2	B2	C1	10,16
A1	B1	C2	7,15
A2	B1	C2	8,11
A1	B2	C2	4,9
A2	B2	C2	12,14

Gültigkeit (Threats to Validity)



168 / 504

Definition

Conclusion Validity: es gibt einen signifikanten statistischen Zusammenhang von Treatment und Resultat

Definition

Internal Validity: der beobachtete Zusammenhang zwischen Treatment und Resultat ist kausal

Definition

Construct Validity:

1. Treatment reflektiert Construct of Cause
2. Outcome reflektiert den Effekt

Definition

External Validity: das Ergebnis ist auch außerhalb der Studie anwendbar (für andere Personen, Orte, Zeiten etc.)

Internal Validity

- **Störvariable:** zusätzliche unkontrollierte Variable ändert sich systematisch mit den unabhängigen Variablen
z.B. Pair-Programmer-Teams bestehen nur aus Entwicklern mit großer Erfahrung
- **Historie/Zeit:** äußere Ereignisse beeinflussen Messung
z.B. Pair-Programmer arbeiten morgens, Einzel-Programmierer nachmittags
- **Sättigung:** interne (physische oder psychische) Änderung der Subjects
z.B. Ermüdung

169 / 504

Internal Validity

- **Wiederholtes Testen:** frühe Treatments beeinflussen nachfolgende Treatments
z.B. Lerneffekte
- **Instrumentierung:** Veränderung der Art der Messung
z.B. Fehler bei der Messung mit späterer Korrektur
- **Regression zur Mitte:** bei zwei in irgendeiner Weise verbundenen Messungen gehen extreme Abweichungen bei einer der beiden Messungen im Durchschnitt mit weniger extremen Abweichungen bei der anderen Messung einher
z.B. Körpergröße von Vätern und Söhnen

170 / 504

Die Regression zur Mitte ist ein Begriff der Statistik; er beschreibt das Phänomen, dass bei zwei in irgendeiner Weise verbundenen Messungen, z. B. Körpergröße eines Vaters und seines Sohnes, extreme Abweichungen bei einer der beiden Messungen im Durchschnitt mit weniger extremen Abweichungen bei der anderen Messung einhergehen – im Beispiel also haben sehr große Väter im Durchschnitt Söhne die weniger groß sind (die aber im Durchschnitt trotzdem noch größer sind als der Durchschnitt der Bevölkerung), oder umgekehrt: sehr große Söhne haben im Durchschnitt Väter die weniger groß sind.

– Quelle: Wikipedia

Internal Validity

- **Selektion:** keine zufällige Zuordnung oder Zuordnung ist zufällig ungünstig
z.B. die ersten 20 Freiwilligen werden der Pair-Programming-Gruppe zugeordnet
- **Selektionsinteraktion:** Selektions-Threat wird mit anderem internal Threat kombiniert
z.B. eine Gruppe wird von Programmierern aus Abteilung X dominiert (Selektions-Threat) und Abteilung X hat am Vorabend eine wilde Party (Historie-Threat)

Internal Validity

- **Experimentelle Mortalität:** Unterschiede in der Ausscheiderate über die unterschiedlichen experimentellen Bedingungen
z.B. in der Pair-Programming-Gruppe fallen mehrere Subjects aus, was die Gruppenzusammensetzung beeinflussen kann
- **Experimentatoreinfluss:** Erwartungen der Durchführenden oder Subjects können Resultat beeinflussen
z.B. Gruppen werden unbewusst verschieden behandelt oder Subjects versuchen, das vermeintlich gewünschte Ergebnis zu erreichen

172 / 504

External Validity

- **Repräsentanz:** Subjects sind nicht repräsentativ
- **Eignung-Treatment-Interaktion:** Sample hat Eigenschaften, die mit der unabhängigen Variablen interagieren
z.B. Subjects sind hochmotivierte Freiwillige.
- **Situation:** situationsbedingte (zeitliche/örtliche) Eigenheiten beeinflussen Ergebnis
z.B. große Displays im Pair-Programming-Experiment
z.B. Tests finden immer nur morgens statt
- **Reaktivität:** Effekt ist auf die Beobachtung der Situation zurückzuführen
z.B. Programmierer sind während des Experiments konzentrierter

173 / 504

Wiederholungs- und Vertiefungsfragen I

- Welche Arten von empirischen Untersuchungsmethoden gibt es generell? Was sind ihre Vor- und Nachteile?
- Gegeben sei folgendes Szenario [...] Welche Art von empirischer Untersuchungsmethode würden Sie einschlagen und warum?
- Was sind die wesentlichen Bestandteile eines kontrollierten Experiments? Erläutern Sie diese an einer konkreten Fragestellung.
- Wo liegen die Grenzen empirischer Forschung?
- Was ist der Zusammenhang eines kontrollierten Experiments und einer Theorie?
- Was unterscheidet ein Experiment von einer bloßen Beobachtung?

174 / 504

Wiederholungs- und Vertiefungsfragen II

- Erläutern Sie die Begriffe *abhängige und unabhängige Variablen, Faktoren, Levels, Treatment, Objekt und Subjekt, Test* und *experimenteller Fehler*.
- Erläutern Sie verschiedene Sampling-Methoden und diskutieren Sie ihre Vor- und Nachteile.
- Erläutern Sie die generellen Prinzipien *Randomization, Blocking, Balancing* und *Replication* eines kontrollierten Experiments.
- Beschreiben Sie verschiedene Versuchsaufbauten in Abhängigkeit der Anzahl der Subjekte und Objekte eines Experiments.
- Beschreiben Sie die Arten von Validitäten einer empirischen Beobachtung. Geben Sie zu jeder Art potentielle Probleme an.

175 / 504

Statistik bei kontrollierten Experimenten

Statistik bei kontrollierten Experimenten

- Hypothesen und Stichproben
- Verteilungen
- Experimente mit einem Sample
- Experimente mit zwei Samples
- Verteilungsfreier U-Test
- Wiederholungsfragen

176 / 504

Hypothese und statistischer Test

Definition

Statistische Hypothese: Aussage über eine statistische Population, die man auf Basis beobachteter Daten zu bestätigen oder zu falsifizieren versucht.

Hypothese:

„Die durchschnittliche Länge von Methoden in Java ist größer als 50 [loc]“

177 / 504

Vorgehen

- ① Nimm an, dass die zu testende Hypothese wahr ist.
- ② Untersuche die Konsequenzen dieser Annahme in Bezug auf die Sampling-Verteilung, die von der Wahrheit der Hypothese abhängt.
 - Falls die beobachteten Daten eine große Eintrittswahrscheinlichkeit haben, ist die Hypothese bestätigt.
 - Falls die beobachteten Daten eine sehr kleine Eintrittswahrscheinlichkeit haben, gilt die Hypothese als widerlegt.
 - **Signifikanzniveau** α legt die Wahrscheinlichkeit fest, ab der die Hypothese als widerlegt betrachtet wird (konkreter Schwellwert: **kritischer Wert**).

Konvention: $\alpha = 0,05$ oder $\alpha = 0,01$

Nullhypothese und alternative Hypothese

Definition

Nullhypothese H_0 : die zu testende Hypothese.

Alternative Hypothese H_1 : die Gegenthese zu H_0 .

Meist: H_1 ist das, woran der Experimentator wirklich glaubt.

→ Experiment soll H_0 widerlegen.

179 / 504

Gerichtete und ungerichtete Hypothese

Definition

Ungerichtete Alternativhypothese: Nullhypothese postuliert keinerlei Effekt.

Gerichtete Alternativhypothese: Nullhypothese postuliert keinen oder gegengerichteten Effekt.

Beispiel ungerichtete Alternativhypothese:

- H_1 = Pair-Programming und Single-Programming unterscheiden sich in Qualität.
- H_0 = Pair-Programming und Single-Programming liefern gleiche Qualität.

Beispiel gerichtete Alternativhypothese:

- H_1 = Pair-Programming liefert bessere Qualität als Single-Programming.
- H_0 = Pair-Programming liefert gleiche oder schlechtere Qualität als Single-Programming.

180 / 504

Die Nullhypothese drückt inhaltlich immer aus, dass Unterschiede, Zusammenhänge, Veränderungen oder besondere Effekte in der interessierenden Population überhaupt nicht und/oder nicht in der erwarteten Richtung auftreten. Im Falle einer *ungerichteten* Forschungs- bzw. Alternativhypothese postuliert die Nullhypothese keinerlei Effekt. Im Falle einer *gerichteten* Alternativhypothese geht die Nullhypothese von keinem oder einem gegengerichteten Effekt aus.

– Bortz und Döring (2006)

Hypothesen und Stichproben

Sample = Population $\Rightarrow$ absolute Wahrheit

Sample $\subset$ Population $\Rightarrow$?

Problem:

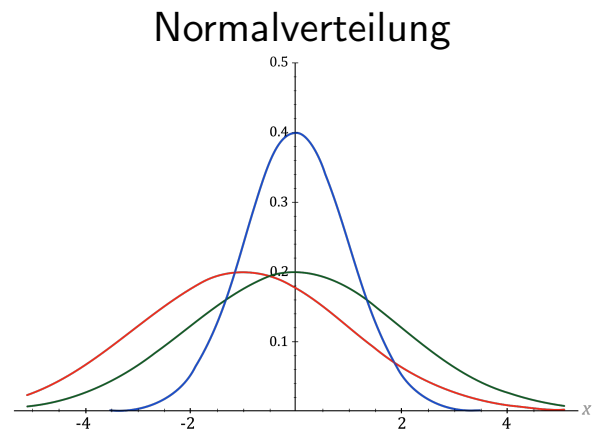
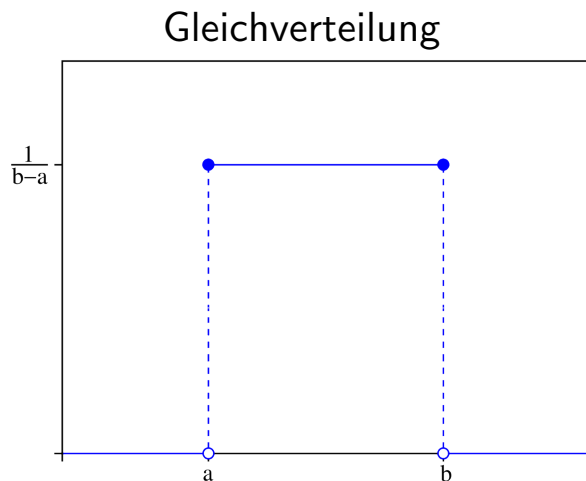
- Jede Hypothesenüberprüfung liefert statistischen Kennwert (z.B. Durchschnitt) für ein bestimmtes Sample.
 - Wiederholung mit anderen Subjects/Objects liefert wahrscheinlich nicht exakt denselben Kennwert.
- Kennwert ist Zufallsvariable¹
- Feststellung, ob Kennwert extrem oder typisch ist, ist ohne Kenntnis der Verteilung der Zufallsvariablen unmöglich.

¹Funktion, die den Ergebnissen eines Zufallsexperiments Werte (so genannte Realisationen) zuordnet.

Verteilungen

Definition

Verteilung einer Variablen: beschreibt, welche Werte die Variable annehmen kann und wie oft sie das tut.



182 / 504

Häufige Kennwerte einer Verteilungen

Gegeben: n Datenpunkte $x_1, x_2, \dots, x_n$ einer Variablen X .

Durchschnitt oder arithmetisches Mittel $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^n x_i$

Varianz $s_x^2 = \frac{1}{n-1} \cdot \sum_{i=1}^n (x_i - \bar{x})^2$

Standardabweichung $s_x = \sqrt{s_x^2}$

183 / 504

Varianz und Freiheitsgrad

$$\text{Varianz } s_x^2 = \frac{1}{n-1} \cdot \sum_{i=1}^n (x_i - \bar{x})^2$$

Warum Durchschnitt mit $\frac{1}{n-1}$?

$$\sum_{i=1}^n (x_i - \bar{x}) = 0$$

- $(x_n - \bar{x})$ kann berechnet werden, wenn $x_1, x_2, \dots, x_{n-1}$ bekannt sind
- nur $n - 1$ Summanden in $\sum_{i=1}^n (x_i - \bar{x})^2$ können frei variieren
- $n - 1$ ist der **Freiheitsgrad**

184 / 504

Verteilung von Population und Sample

H_1 : Durchschnittliche Länge von Java-Methoden $\mu > 50$

H_0 : Durchschnittliche Länge von Java-Methoden $\mu \leq 50$

Gegeben:

- Populations-Verteilung: Kennwerteverteilung der Population P mit Durchschnitt μ und Standardabweichung σ
- Sample-Verteilung: Kennwerteverteilung der Stichproben X mit Durchschnitt $\bar{x}$ und Standardabweichung $s_{\bar{x}}$

Annahmen:

- σ ist bekannt
- P hat Normalverteilung

Daraus folgt: X ist normalverteilt mit $\bar{x} = \mu$ und $s_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$.

185 / 504

Verteilung von Population und Sample

Warum gilt: $\bar{x} = \mu$?

Sample-Größe ist n .

Jeder beobachtete Wert x_i ($1 \leq i \leq n$) ist eine Messung von einem zufällig ausgewählten Element aus P .

→ Jede Einzelmessung ist eine Zufallsvariable X_i , deren Verteilung der von P entspricht.

$$\bar{x} = \frac{1}{n} \cdot (X_1 + X_2 + \dots X_n)$$

Wenn μ der Durchschnitt von P ist, dann ist μ der Durchschnitt der Verteilung jeder Beobachtung X_i .

$$\mu_{\bar{x}} = \frac{1}{n} \cdot (\mu_{X_1} + \mu_{X_2} + \dots \mu_{X_n}) = \frac{1}{n} \cdot (\mu + \mu + \dots \mu) = \mu$$

186 / 504

Verteilung von Population und Sample

Warum gilt: $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$?

Regeln für Varianzen (a, b sind Konstante, X, Y Zufallsvariablen):

$$\sigma_{a+bX}^2 = b^2 \sigma_X^2$$

$$\sigma_{X+Y}^2 = \sigma_X^2 + \sigma_Y^2$$

Damit:

$$\sigma_{\bar{x}}^2 = \sigma_{\frac{1}{n} \cdot (X_1 + X_2 + \dots X_n)}^2 = \left(\frac{1}{n}\right)^2 \cdot (\sigma_{X_1}^2 + \sigma_{X_2}^2 + \dots \sigma_{X_n}^2)$$

Weil jede Einzelbeobachtung X_i aus P stammt, gilt $\sigma_{X_i}^2 = \sigma^2$ und damit:

$$\sigma_{\bar{x}}^2 = \left(\frac{1}{n}\right)^2 \cdot (\sigma^2 + \sigma^2 + \dots \sigma^2) = \frac{\sigma^2}{n} \text{ und } \sigma_{\bar{x}} = \sqrt{\sigma_{\bar{x}}^2} = \frac{\sigma}{\sqrt{n}}$$

187 / 504

Verteilung von Population und Sample

H_1 : Durchschnittliche Länge von Java-Methoden $\mu > 50$

H_0 : Durchschnittliche Länge von Java-Methoden $\mu \leq 50$

Gegeben:

- Populations-Verteilung: Kennwerteverteilung der Population P mit Durchschnitt μ und Standardabweichung σ
- Sample-Verteilung: Kennwerteverteilung der Stichproben X mit Durchschnitt $\bar{x}$ und Standardabweichung $s_{\bar{x}}$

Annahmen:

- σ ist bekannt
- P hat Normalverteilung

Daraus folgt: X ist normalverteilt mit $\bar{x} = \mu$ und $s_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$.

188 / 504

Beispiel

$H_0 : \mu = 50$.

Sei tatsächlich beobachteter Wert (Messung) für $\bar{x} = 54$ mit $\sigma = 10$ und Sample-Größe $n = 25$.

Passt das noch zu H_0 mit Signifikanzniveau $\alpha = 0,01$?

$\bar{x}$ ist normalverteilt mit $\mu = 50$ und $\sigma_{\bar{x}}^2 = \frac{10^2}{\sqrt{25}} = 2$: $N(50, 2)$

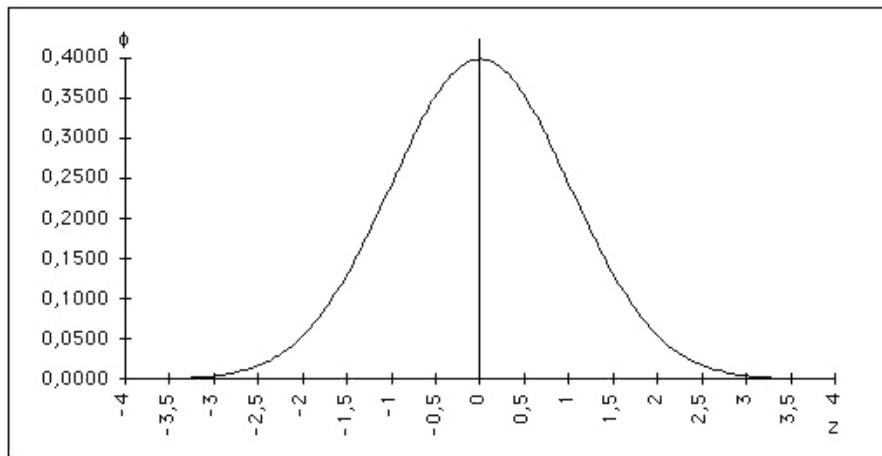
Die Standardnormalverteilung $N(0, 1)$ ist tabelliert. Mit **z-Transformation** kann jede Normalverteilung auf $N(0, 1)$ zurückgeführt werden:

$$z_{\bar{x}} = \frac{\bar{x} - \mu}{\sigma_{\bar{x}}}$$

189 / 504

Beispiel

Wahrscheinlichkeit, einen Wert $z_{\bar{x}} = \frac{54-50}{\sqrt{2}} \approx 1,41$ oder größer in $N(0, 1)$ zu finden = Flächeninhalt zwischen 1,41 und ∞ in $N(0, 1)$



Laut Tabelle für $N(0, 1)$: $1 - 0,9207 = 0,0793 > 0,01 = \alpha$.

→ H_0 wird nicht abgelehnt

190 / 504

Wir fragen nach der Wahrscheinlichkeit, mit der Stichprobenergebnisse auftreten können, wenn die Nullhypothese gilt. Wir betrachten nur diejenigen Ergebnisse, die bei Gültigkeit der Nullhypothese höchstens mit einer Wahrscheinlichkeit von α (z.B. 1 % oder 5 %) vorkommen. Gehört das gefundene Stichprobenergebnis zu diesen Ergebnissen, ist das Stichprobenergebnis „praktisch“ nicht mit der Nullhypothese vereinbar. Wir entscheiden uns deshalb dafür, die Nullhypothese abzulehnen und akzeptieren die Alternativhypothese als Erklärung für unser Untersuchungsergebnis.

– Bortz und Döring (2006)

Laut Tabellierung von $N(0, 1)$ ist die Fläche von $(-\infty, 1,41] = 0,9207$.

Beispieluntersuchung

Hypothese: „Pair-Programming unterscheidet sich in durchschnittlicher Fehlerdichte $\frac{\#Fehler}{LOC}$ von Single-Programming.“

Design:

- Object: Anforderungsspezifikation
- Subjects: 31 professionelle Entwickler
- Blocking:
 - Treatment X: eine Gruppe (10×2) wendet Pair-Programming an
 - Treatment Y: eine Gruppe (11×1) wendet Pair-Programming nicht an

→ ein Faktor, zwei Treatments

191 / 504

Experiment mit zwei Samples: t-Test

Gegeben: Zwei unabhängige Samples:

$X = x_1, x_2, \dots, x_n$ mit Durchschnitt $\bar{x}$ und Varianz s_x^2
 $Y = y_1, y_2, \dots, y_m$ mit Durchschnitt $\bar{y}$ und Varianz s_y^2

H_0 : Mittelwerte von X und Y sind gleich: $\mu_x - \mu_y = 0$.

Annahmen:

Population zu X ist normalverteilt mit Durchschnitt μ_x und Varianz σ_x^2 ,

Population zu Y ist normalverteilt mit Durchschnitt μ_y und Varianz σ_y^2

und $\sigma_x^2 = \sigma_y^2$.

Aber: Varianz σ_x^2 von X und Y ist unbekannt.

192 / 504

Experiment mit zwei Samples: t-Test

Mittelwert von $\bar{x} - \bar{y}$ ist gleich dem Mittelwert von $\mu_x - \mu_y$.

Folgt aus:

- Additionsregel für Mittelwerte
- und Mittelwert von jedem Messwert $\bar{x}$ ist der Mittelwert seiner Population μ

193 / 504

Experiment mit zwei Samples: t-Test

Varianz von $\bar{x} - \bar{y}$ ist:

$$\frac{\sigma_x^2}{n} + \frac{\sigma_y^2}{m}$$

Folgt aus Additionsregel für Varianzen.

194 / 504

Experiment mit zwei Samples: t-Test

Satz: Wenn beide Populationen normalverteilt sind, dann ist die Verteilung von $\bar{x} - \bar{y}$ auch normalverteilt.

z-Transformation einer Zufallsvariablen hat Standardnormalverteilung $N(0, 1)$:

$$z = \frac{(\bar{x} - \bar{y}) - (\mu_x - \mu_y)}{\sqrt{\frac{\sigma_x^2}{n} + \frac{\sigma_y^2}{m}}}$$

195 / 504

Experiment mit zwei Samples: t-Test

Annahme war: beide Populationen haben gleiche Varianz

$$\sigma_\epsilon^2 = \sigma_x^2 = \sigma_y^2$$

Varianz von σ_ϵ^2 kann geschätzt werden durch zusammengelegte Varianzen s_p^2 als gewichteter Durchschnitt:

$$s_p^2 = \frac{(n-1)s_x^2 + (m-1)s_y^2}{(n-1) + (m-1)}$$

Damit ist z-Transformation für die Schätzung:

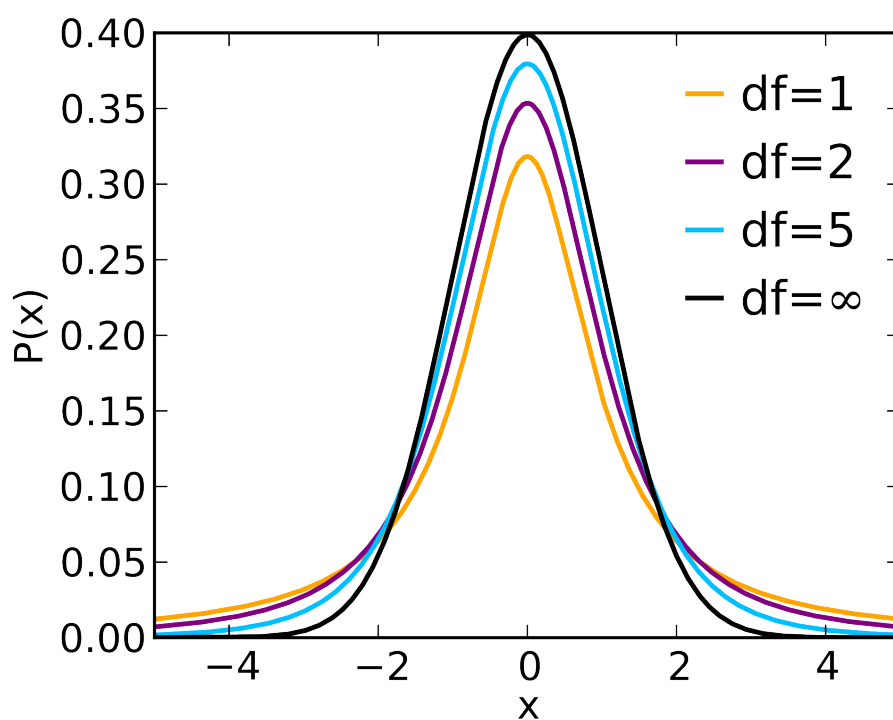
$$t = \frac{(\bar{x} - \bar{y}) - (\mu_x - \mu_y)}{\sqrt{\frac{s_p^2}{n} + \frac{s_p^2}{m}}}$$

→ t folgt Students t-Verteilung mit $(n-1) + (m-1) = n + m - 2$ Freiheitsgraden (df)

196 / 504

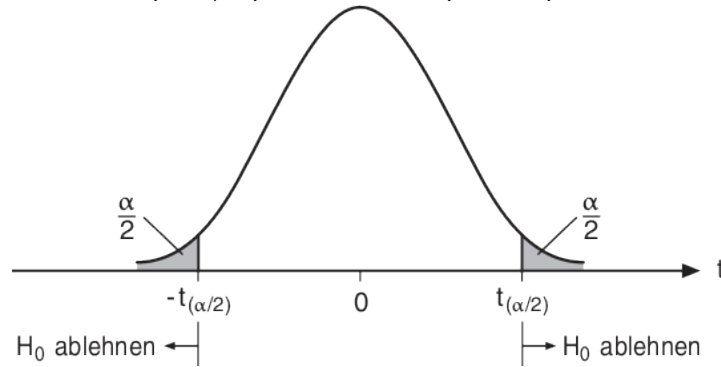
Die Annahme ist, dass die Samples beide eine gemeinsame homogene Varianz haben. Dann kann diese geschätzt werden, indem die Informationen beider Samples gebündelt werden. Die Schätzung ist dann der gewichtete Durchschnitt der einzelnen Varianzen beider Sample-Varianzen. Die Gewichte hierfür sind die jeweiligen Freiheitsgrade $n - 1$ und $m - 1$. S_p ist dann die gebündelte Varianz. Der Freiheitsgrad von S_p ist $(n - 1) + (m - 1) = n + m - 2$.

Students t-Verteilung (df = Freiheitsgrad)

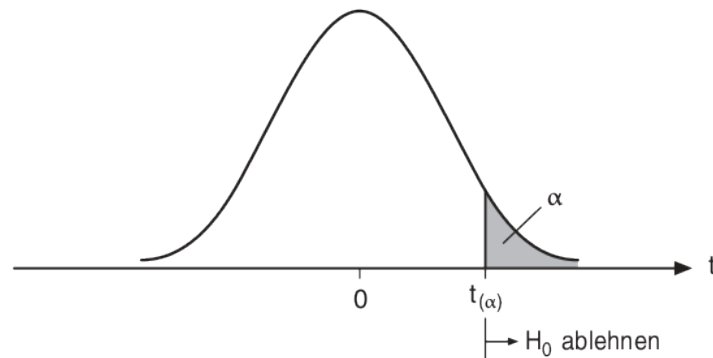


Students t-Verteilung

Ungerichtete $H_1 \equiv \mu_1 \neq \mu_2 \wedge H_0 \equiv \mu_1 = \mu_2 \rightarrow$ zweiseitiger Test



Gerichtete $H_1 \equiv \mu_1 > \mu_2 \wedge H_0 \equiv \mu_1 \leq \mu_2 \rightarrow$ einseitiger Test



198 / 504

Ungerichtete Alternativhypothese $H_1 \equiv \mu_1 \neq \mu_2$: Nullhypothese postuliert keinerlei Effekt $H_0 \equiv \mu_1 = \mu_2$.

Gerichtete Alternativhypothese $H_1 \equiv \mu_1 > \mu_2$: Nullhypothese postuliert keinen oder gegengerichteten Effekt $H_0 \equiv \mu_1 \leq \mu_2$.

Gerichtete Hypothesen werden anhand der Verteilung über einseitige und ungerichtete Hypothesen über zweiseitige Tests geprüft. Bei einem zweiseitigen Test markieren die Werte $t(\alpha/2)$ und $-t(\alpha/2)$ diejenigen t-Werte einer t-Verteilung, die von den Extremen der Verteilungsfläche jeweils $\alpha/2\%$ abschneiden.

Zusammenfassung des Vorgehens beim t-Test

- Eingabe: Zwei unabhängige Samples $x_1, x_2, \dots, x_n$ und $y_1, y_2, \dots, y_m$
- Annahme: Populationen zu X und Y sind normalverteilt und haben gleiche Varianz
- H_0 : Mittelwerte von X und Y sind gleich: $\mu_x - \mu_y = 0$
- Transformation von H_0 : $t_0 = \frac{\bar{x} - \bar{y}}{s_p \times \sqrt{\frac{1}{n} + \frac{1}{m}}}$

wobei $s_p = \sqrt{\frac{(n-1)s_x^2 + (m-1)s_y^2}{(n-1) + (m-1)}}$

und s_x^2 und s_y^2 sind die individuellen Sample-Varianzen

- t_0 folgt bei Gültigkeit von H_0 einer t-Verteilung mit $n + m - 2$ Freiheitsgraden
- Kriterium (zweiseitig, mit Signifikanzniveau α):
 H_0 ablehnen, wenn $|t_0| > t_{\alpha/2, n+m-2}$

199 / 504

Beispielmessungen

Treatment X = Pair-Programming, Treatment Y = kein Pair-Programming

i	Treatment X: x_i	Treatment Y: y_i
1	3,24	3,44
2	2,71	4,97
3	2,84	4,76
4	1,85	4,96
5	3,22	4,10
6	3,48	3,05
7	2,68	4,09
8	4,30	3,69
9	2,49	4,21
10	1,54	4,40
11		3,49
	n=10	m=11
	$\bar{x} = 2,835$	$\bar{y} = 4,1055$
	$S_x^2 = 0,6312$	$S_y^2 = 0,4112$

200 / 504

Beispielauswertung mit t-Test

$$\begin{aligned} s_p &= \sqrt{\frac{(n-1)s_x^2 + (m-1)s_y^2}{(n-1) + (m-1)}} \\ &= \sqrt{\frac{(10-1) \cdot 0,6312 + (11-1) \cdot 0,4112}{(10-1) + (11-1)}} = -0,564 \end{aligned}$$

$$\begin{aligned} t_0 &= \frac{\bar{x} - \bar{y}}{s_p \times \sqrt{\frac{1}{n} + \frac{1}{m}}} \\ &= \frac{2,835 - 4,1055}{-0,564 \times \sqrt{\frac{1}{10} + \frac{1}{11}}} = -5,642 \end{aligned}$$

Freiheitsgrade: $df = 10 + 11 - 2 = 19$

$\rightarrow t_{\alpha/2, n+m-2} = t_{0,05/2, 10+11-2} = 2,093$

$|t_0| = 5,642 > t_{0,05/2, 10+11-2} = 2,093 \rightarrow H_0$ ablehnen

Exakter U-Test von Mann-Whitney

Gegeben: zwei unabhängige Samples $x_1, x_2, \dots, x_n$ und $y_1, y_2, \dots, y_m$ mit Ordinalskalenniveau.

Annahme: Beide Samples stammen von Populationen mit der gleichen Verteilung.

Keine Annahme über diese Verteilung.

- ① Daten beider Samples werden vereinigt und geordnet.
- ② Jeder Wert x_i wird mit jedem Wert y_j verglichen:
 - G_i = Anzahl der $y_j < x_i$
 - L_i = Anzahl der $y_j > x_i$
- ③ Summiere:
 - $G = \sum_{1 \leq i \leq n} G_i$
 - $L = \sum_{1 \leq i \leq n} L_i$
 - $U = \min(\bar{L}, G)$

Gruppe	x_i bzw. y_i	G_i	L_i
X	1.54	11	0
X	1.85	11	0
X	2.49	11	0
X	2.68	11	0
X	2.71	11	0
X	2.84	11	0
Y	3.05		
X	3.22	10	1
X	3.24	10	1
Y	3.44		
X	3.48	9	2
Y	3.49		
Y	3.69		
Y	4.09		
Y	4.10		
Y	4.21		
X	4.30	4	7
Y	4.40		
Y	4.76		
Y	4.96		
Y	4.97		
Σ		99	11

203 / 504

Signifikanztest zum exakten U-Test von Mann-Whitney

- Es gibt $\binom{n+m}{m} = \binom{n+m}{n}$ mögliche Rangfolgen.
- Erwartungswert für U bei H_0 : $\mu_U = (n + m)/2$.
- Je weiter beobachtetes U vom Erwartungswert abweicht, desto unwahrscheinlicher ist H_0 .
- Einseitiger Test:
 - Z_U = Anzahl möglicher Kombinationen, die einen U-Wert liefern, der nicht größer als U ist.
 - $P = Z_U / \binom{n+m}{m}$
- Zweiseitiger Test:
 - Z'_U = Anzahl möglicher Kombinationen, die einen U-Wert liefern, der nicht kleiner als $\max(L, G)$ ist.
 - $P = (Z_U + Z'_U) / \binom{n+m}{m}$
- Lehne H_0 ab, wenn $P \leq \alpha$.
- Kritischer Wert (der zur Ablehnung von H_0 führt) kann in Tabelle des U-Tests für kleine Samples nachgeschlagen werden.

Im Beispiel: kritischer Wert = 26 für $\alpha = 0,05$

→ H_0 wird abgelehnt wegen $U < 26$

204 / 504

Tabellen für den kritischen Wert bei gegebenem Signifikanzniveau für den U-Test lassen sich im Web finden, indem man nach den Stichwörtern „table u test“ sucht. Z.B.: math.usask.ca/~laverty/S245/Tables/wmw.pdf
Es wird vorausgesetzt, dass keine identischen Messwerte („Bindungen“ oder „Rangbindungen“) auftreten. Falls Bindungen vorhanden sind, werden den Werten die mittleren Rangzahlen zugewiesen.

Weiterführende Literatur

Empirische Methoden

- Endres und Rombach (2003) beschreiben wesentliche empirische Kenntnisse in der Software-Technik und brechen eine Lanze für die empirische Forschung in diesem Gebiet.
- Lienert (1973) beschreibt verteilungsfreie (nicht-parametrische) statistische Tests
- Prechelt (2001) beschreibt empirische Methoden in der Softwaretechnik (deutschsprachig, leider vergriffen und wird nicht mehr neu aufgelegt)
- Wohlin u. a. (2000) beschreibt empirische Methoden in der Softwaretechnik
- Christensen (2007) beschreibt experimentelle Methoden im Allgemeinen

Weiterführende Literatur

Statistik in der Empirie

- Bortz u. a. (2008) beschreiben experimentelle Designs und ihre statistischen (nicht-parametrischen, d.h. verteilungsfreien) Auswertungen
- Winner u. a. (1991) beschreiben experimentelle Designs und ihre statistischen (parametrischen) Auswertungen
- Moore u. a. (2009) geben eine allgemeine Einführung in Statistik

206 / 504

Wiederholungs- und Vertiefungsfragen

- Was ist eine statistische Hypothese? Wie wird sie überprüft und welche Rolle spielt dabei das Signifikanzniveau (der kritische Wert)?
- Welche Arten von Hypothesen gibt es?
- Mit welchen Maßen werden Population und Sample meist statistisch charakterisiert?
- Was versteht man unter einem parametrischen bzw. nichtparametrischen Test?
- Erläutern Sie das Prinzip des t-Tests.
- Erläutern Sie das Prinzip des exakten U-Tests von Mann-Whitney.

207 / 504

Kosten- und Aufwandsschätzung

Kosten- und Aufwandsschätzung

Kostenschätzung

Function-Points

Object-Points

COCOMO

Wiederholungsfragen

208 / 504

Kostenschätzung

Wichtige Fragen vor einer Software-Entwicklung:

- Wie hoch wird der Aufwand sein?
- Wie lange wird die Entwicklung dauern?
- Wie viele Leute werden benötigt?

Frühzeitige Beantwortung wichtig für:

- Kalkulation und Angebot
- Personalplanung
- Entscheidung „make or buy“
- Nachkalkulation

210 / 504

Schätzgegenstand

- **Kosten:** was zu bezahlen ist
 - Aufwand $\times$ Kosten pro Aufwandseinheit
 - Schätzen: 50 Euro/DLOC (Delivered Line of Code)
- **Dauer:** wie lange man warten muss
 - Aufwand in PM/Anzahl Mitarbeiter
(reell nicht-linearer Zusammenhang laut Brooks (1995))
 - Parkinsons Gesetz
- **Aufwand:** was man an Ressourcen braucht
 - Umfang + Risiko + Management + ...

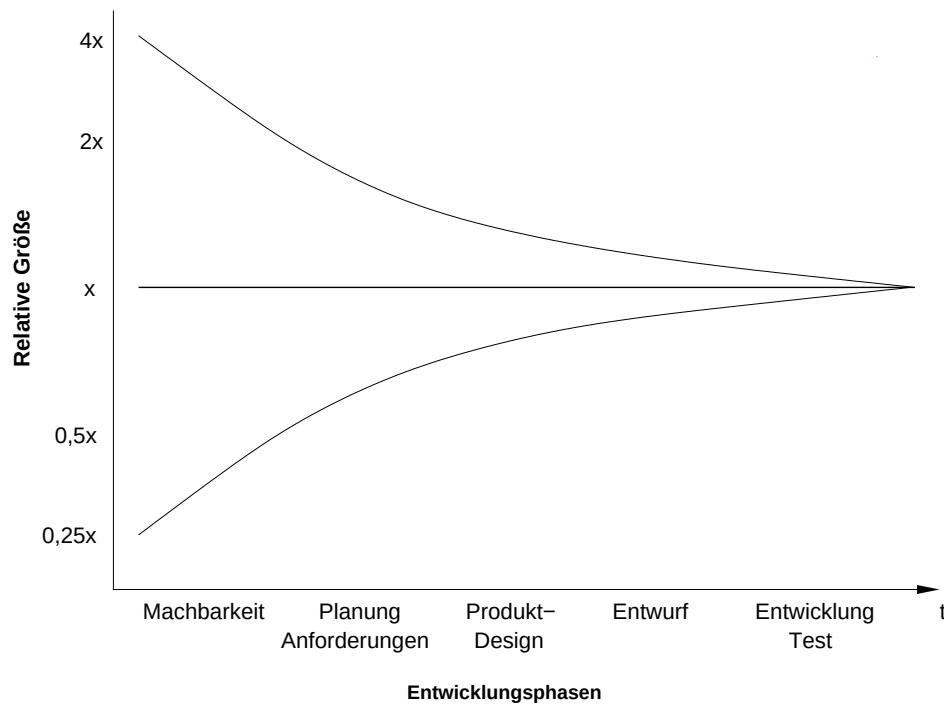
- Expertenschätzung
- Analogiemethode
- Top-Down-Schätzung
- Bottom-Up-Schätzung
- Pricing-to-Win/Festpreis
- Algorithmische Schätzung
 - COCOMO: Anzahl Codezeilen
 - Schätzung auf Basis von Function-Points: Ein- und Ausgaben

212 / 504

- Expertenschätzung: mehrere Experten fragen; Abweichungen diskutieren, bis Konsens erreicht
→ Schätzpoker bei Scrum
- Analogie: dauert so lange wie beim ähnlichen Projekt X; Bezug auf historische Daten eines ähnlichen Projekts
- Top-Down-Schätzung: Ableitung aus globalen Größen
 - z.B. Aufwand steht fest, daraus Umfang ableiten
- Bottom-Up-Schätzung: Dekomposition und Schätzung der einzelnen Komponenten sowie deren Integrationsaufwand

Pricing-To-Win: Preis wird vereinbart; im Zuge des Projekts einigt man sich auf Funktionsumfang (eigentlich keine Schätzung).

Berechnung aus früh bekannten Größen; statistisches Kostenmodell aus Vergangenheitsdaten wird erstellt; Modell wird zur Vorhersage benutzt.



213 / 504

Function-Point-Methode

Benötigt für frühe Kostenschätzung: Maß für den Umfang der Software.

Function-Points:

- Messen des **funktionalen Umfangs**² einer zu erstellenden Software aus Benutzersicht
- Eingabe: Lastenheft

Einsatz:

- Umrechnung des Umfangs in personellen Aufwand
- Vergleich und Bewertung von Systemen
- Messung von Produktivität, Benchmarking

²nicht des Aufwands

Historische Entwicklung der Function-Point-Methode

- 1979: erste Veröffentlichung: Alan J. Albrecht (1979) (IBM)
- 1985: Veröffentlichung der „IBM-Kurve“ (IBM 1985): Zusammenhang von Aufwand und Function-Points für 54 Projekte
- 1990: Hype: fast alle großen Unternehmen probieren Function-Point-Analyse aus
- 1995: Abkühlung des Interesses
- 1995-heute: wieder gesteigertes Interesse:
- Heute:
 - zahlreiche Varianten
 - IFPUG Int'l Function Point User Group

217 / 504

- 1979: erste Veröffentlichung: Alan J. Albrecht (1979) (IBM)
- 1985: Veröffentlichung der „IBM-Kurve“ (IBM 1985): Zusammenhang von Aufwand und Function-Points für 54 Projekte
- 1990: Hype: fast alle großen Unternehmen probieren Function-Point-Analyse aus
- 1995: Abkühlung des Interesses
 - Unerfahrenheit der Anwender
 - unrealistische Erwartungen
 - zu wenig Erfahrungsdaten vorhanden
- 1995-heute: wieder gesteigertes Interesse:
 - Benchmarking
 - Wirtschaftlichkeit (Function-Points versus Kosten)
 - Outsourcing
 - Offshoring
- Heute:
 - zahlreiche Varianten
 - IFPUG Int'l Function Point User Group

<http://www.ifpug.com/fpafund.htm>

FAQ: <http://ourworld.compuserve.com/homepages/softcomp/fpfaq.htm>

Function-Point-Methode – Vorgehen

- ① Zähltyp festlegen: Neu-/Weiterentwicklung,
- ② Systemgrenzen festlegen
- ③ Identifizieren der Elementarprozesse und deren Funktionstypen sowie der Datenbestände
- ④ Bewerten des Umfangs der Funktionstypen und Datenbestände
- ⑤ Ermittlung der gewichteten Function Points durch Schätzung von technischen Randbedingungen
- ⑥ Verwendung; z.B. Ermittlung des Aufwands

218 / 504

Elementarprozess

Definition

Elementarprozess: atomare und einzigartige Aktivität des Systems aus Benutzersicht

219 / 504

Definition

Atomaritätsprinzip: Elementarprozess ist kleinste aus fachlicher Sicht sinnvolle, in sich abgeschlossene Aktivität

Bsp.: Erfassung einer Kundenadresse (auch über mehrere Bildschirmmasken)

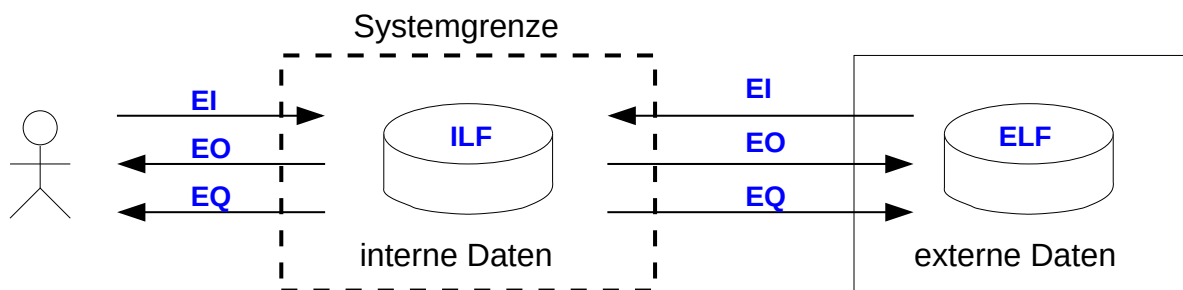
Definition

Einmaligkeitsprinzip: Elementarprozess gilt als einmalig (einzigartig), wenn er durch die ein- oder ausgegebenen Daten oder durch die Verarbeitungslogik unterscheidbar ist (aus Sicht des Anwenders); Unterscheidung durch

- besondere Verarbeitungslogik oder
- verarbeitete Felder der Datenbestände oder
- verarbeitete Datenbestände selbst

Bsp.: Berechnung des Krankenkassenbeitrags einer privaten Versicherung für Neu- bzw. Bestandskunden sind verschiedene Elementarprozesse

FP – Identifizieren der Funktionstypen



Funktionstypen von Elementarprozessen:

- Eingaben: EI (External Input; Eingabe für ILF)
- Ausgaben: EO (External Output; Ausgabe abgeleiteter Daten)
- Abfragen: EQ (External Inquiry; reine Abfrage von Daten ohne Berechnung/Ableitung)

Funktionstypen von Datenbeständen:

- Interner Datenbestand (ILF: Internal Logical File): fachliche Daten, die vom System selbst gepflegt werden (anlegen, ändern, löschen)
- Externer Datenbestand – auch: Referenzdatenbestand– (ELF: External Logical File): fachliche Daten, die vom System nur gelesen werden

Definition

Eingabe: Hauptzweck: internen Datenbestand pflegen oder Systemverhalten ändern, wobei gilt:

- Daten oder Steuerinformationen kommen von außerhalb des Systems
- und mindestens ein ILF wird gepflegt, falls die Daten, die die Systemgrenze überqueren, keine Steuerinformationen sind, die das Systemverhalten verändern
- und mindestens eine der folgenden Bedingungen ist erfüllt (Einzigartigkeit):
 - Verarbeitungslogik ist einzigartig gegenüber anderen Eingaben
 - andere Datenfelder als bei anderen Eingaben werden verwendet
 - andere Datenbestände als bei anderen Eingaben werden verwendet

Definition

Ausgabe:

- Hauptzweck: dem Anwender Informationen präsentieren
- Daten oder Steuerinformationen werden über Systemgrenze geschickt und
- Elementarprozess ist eindeutig (s.o.) und
- und mindestens eine der folgenden Bedingungen ist erfüllt (Abgrenzung zu Abfrage):
 - Verarbeitungslogik enthält mindestens eine mathematische Formel oder Berechnung
 - Verarbeitungslogik pflegt mindestens einen ILF
 - Verarbeitungslogik verändert das Systemverhalten

Definition

Abfrage:

- Hauptzweck: dem Anwender Informationen präsentieren
- Daten oder Steuerinformationen werden über Systemgrenze geschickt und
- Elementarprozess ist eindeutig (s.o.) und
- und **alle** folgende Bedingungen sind erfüllt (Abgrenzung zu Ausgabe):
 - Elementarprozess bezieht Daten oder Steuerinformationen von einem ILF oder ELF
 - Verarbeitungslogik enthält keine mathematische Formel oder Berechnung
 - Verarbeitungslogik erzeugt keine abgeleiteten Daten
 - Verarbeitungslogik pflegt keinen ILF
 - Verarbeitungslogik verändert das Systemverhalten nicht

223 / 504

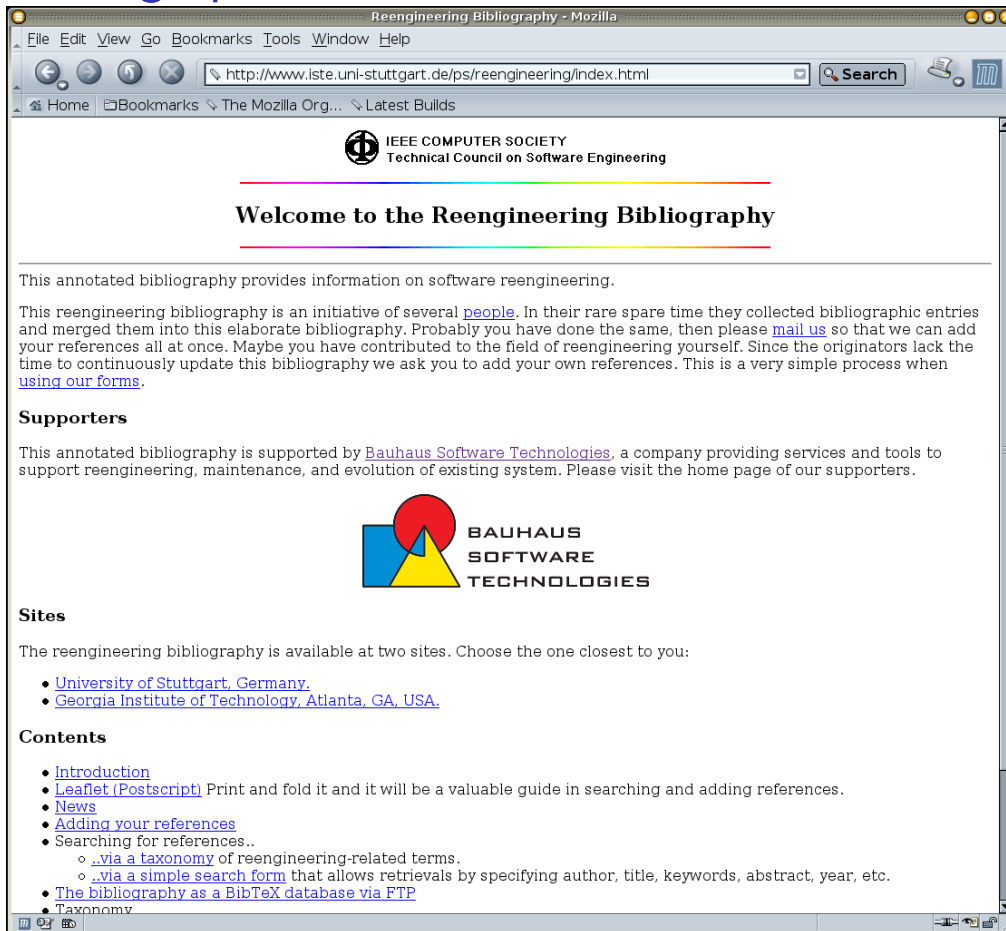
FP – Identifizieren von Funktionstypen

Schlüsselwörter geben Hinweise:

- EI: ablegen/speichern, de-/aktivieren, abbuchen, ändern/editieren/modifizieren/ersetzen, einfügen/hinzufügen, entfernen/löschen, erstellen, konvertieren, update, übertragen
- EO: anzeigen, ausgeben, ansehen, abfragen, suchen/durchsuchen, darstellen, drucken, selektieren, Anfrage, Abfrage, Report
- EQ: abfragen, anzeigen, auswählen, drucken, suchen/durchsuchen, darstellen/zeigen, drop down, extrahieren, finden, holen, selektieren, Ausgabe, Liste, Report

224 / 504

Online-Bibliographie: Startseite



226 / 504

Die Startseite enthält reine Navigationsinformationen. Diese Daten werden nicht gezählt. Weitere Elementarprozesse lassen sich aber auf dieser Seite identifizieren. Wir konzentrieren uns im Folgenden auf die besonders relevanten Elementarprozesse für das Hinzufügen von Referenzen, die Suche mittels Taxonomie und einem Suchformular. Für den Benutzer von außen sichtbar bestehen die internen Datenbestände aus zwei logisch getrennten Inhalten. Zum einen sind das die Referenzen, die selbst in verschiedene Untergruppen zerfallen (Zeitschriftenartikel, Buch, Konferenzbeitrag etc.), zum anderen können wir die Taxonomie als Meta-Daten erkennen. Externe Datenbestände, auf die das System zugreift, sind nicht zu erkennen.

Online-Bibliographie: Neuen Artikel einpflegen

Submitting to the reengineering bibliography - Icedweasel

Home Prev: Submitting References

Submitting a reference to an article in a journal

Please fill out all of the mandatory fields. We appreciate if you fill in the optional fields as well. We are especially interested in the abstract and your classification according to the [taxonomy](#). For further questions we might also need your e-mail address.

[For an example on how to fill out a form, click here.](#)

After you have filled out the form click the "submit" button below.

Your e-mail address:

Mandatory fields

Please use "and" as an author separator as in: E. Chikofsky and James Cross.

Authors:

Title:

Journal:

Year:

Optional fields

Volume:

Number:

Pages:

Month:

228 / 504

Online-Bibliographie: Neuen Artikel einpflegen

Submitting to the reengineering bibliography - Icedweasel

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

If the contents of your reference is available via the web, please add the link information here:

URL:

Abstract:

Keywords:

Your personal note to this reference:

Classification:

The classification of the article is important to support retrieval by users interested in a certain topic. Therefore, we are very much interested in getting the article classified. And if you are the author there is no better expert to do it.

The classification list box below contains a hierarchy of bibliography-related terms. The indentation (stressed by dots) indicates the subclass relation. Choose any item you find appropriate by clicking on it. Clicking twice on the same item unselects it. Note that a parent class is automatically selected if you select one of its subclasses. Furthermore, you can select different classes for the same reference by multiple choices. Please be as specific as possible.

We tried to select terms that are as self-explanatory as possible. If something is not clear just see the [general description of the taxonomy](#).

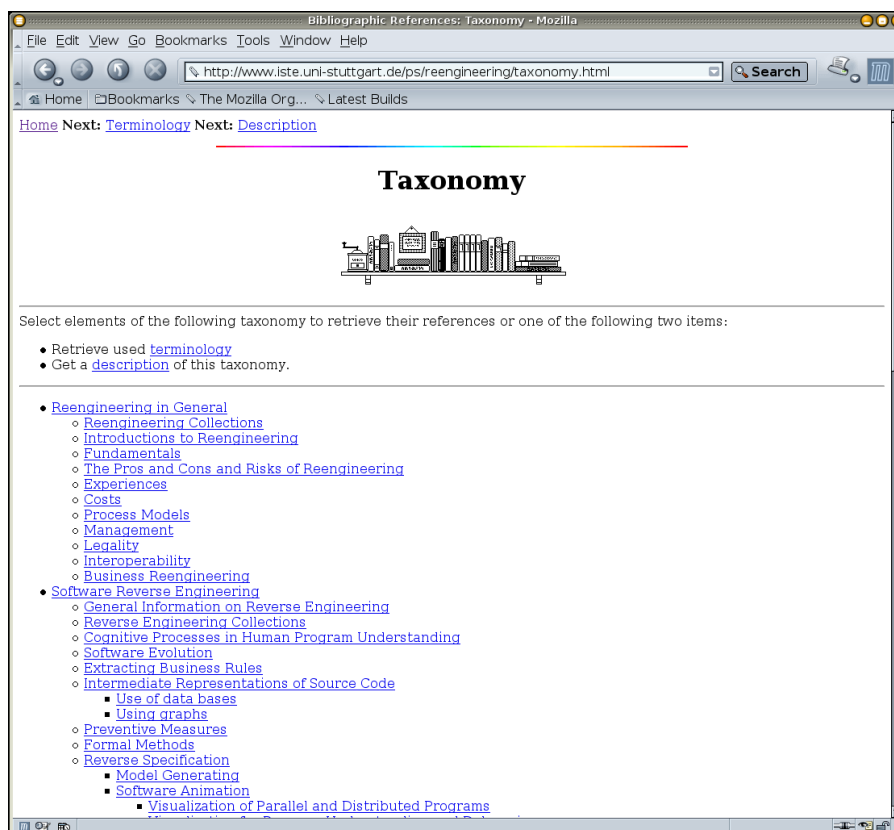
Reengineering_in_General
· Reengineering_Collections
· Introductions_to_Reengineering
· Fundamentals

229 / 504

Mit dieser Seite werden neue Referenzen hinzugefügt. Dieser Elementarprozess hat als Hauptzweck die Eingabe. Die Taxonomie selbst wird nicht betroffen. Einzig der Datenbestand *Referenzen* wird verändert. Über den Link *Classification* kann man sich die Taxonomie anzeigen lassen. Wir betrachten dies als eigenen Elementarprozess *Beschreibung der Taxonomie*. Er unterstützt zwar den anderen Elementarprozess, ist aber nicht Teil von diesem, da der andere Elementarprozess auch ohne ihn vollständig ist. Dieser Elementprozess ist eine klare Abfrage, da nur Daten ohne weitere Verarbeitung angezeigt werden. Ergebnis der Abfrage ist die Taxonomie als Liste (genauer: Baum) von Taxonomieeinträgen.

Insgesamt unterstützt diese Seite also zwei Elementarprozesse.

Online-Bibliographie: Taxonomiesuche



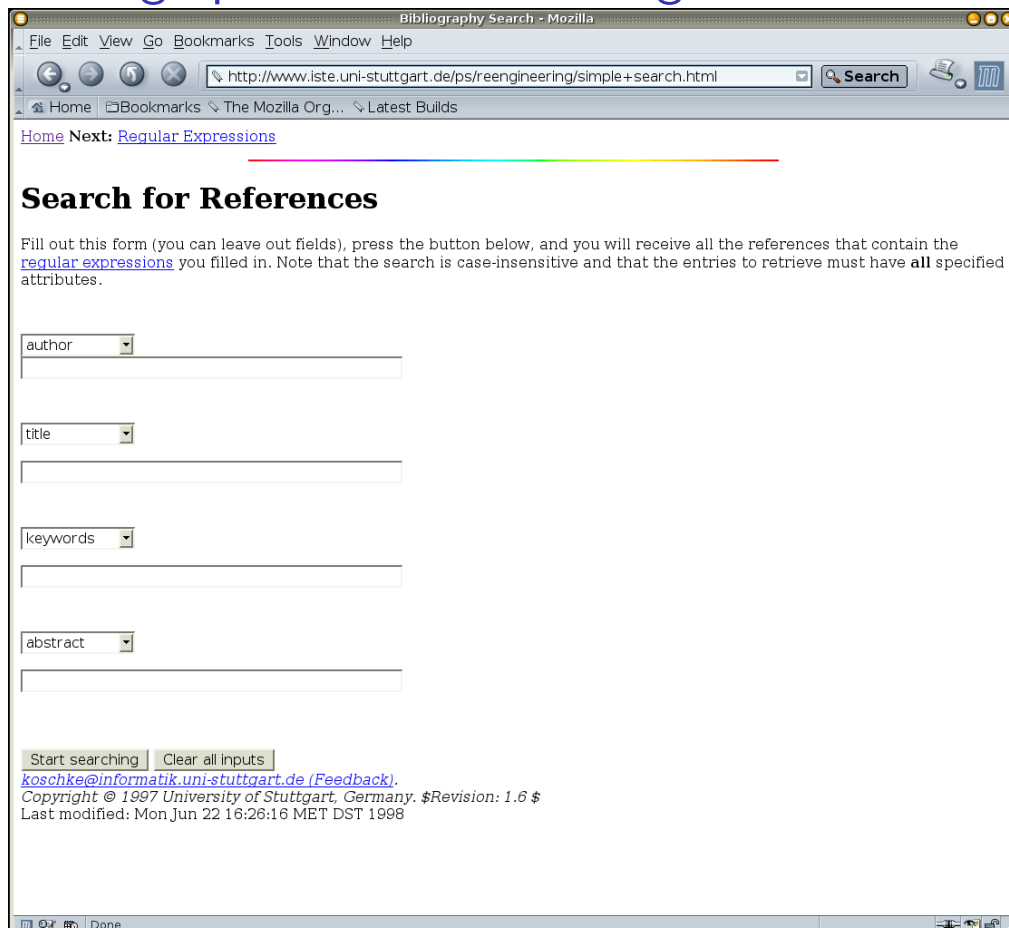
Annahme: Ein BibTeX-Eintrag habe max. 8 Einträge

Auf dieser Seite wird der Elementarprozess *Suche über die Taxonomie* erkennbar. Jeder Taxonomieeintrag stellt einen Link dar, der alle Artikel auflistet, die zu diesem Taxonomiepunkt gehören oder zu Taxonomiepunkten, die von diesem abgeleitet sind. Die Taxonomie selbst wird beim Start dieser Seite vom System bereit gestellt. Letzteres entspricht auf den ersten Blick dem Elementarprozess *Taxonomie anzeigen*. Wir müssen nun bestimmen, ob dieser Elementarprozess von dem weiter oben diskutierten unterscheidbar ist. Er unterscheidet sich nicht durch die Eingabe, die Verarbeitung oder den referenzierten Datenbeständen jedoch in den Ausgabedaten. Während beim ersten Elementarprozess *Beschreibung der Taxonomie* die einzelnen Taxonomiepunkte textuell beschrieben werden, wird uns auf dieser Seite beim Klicken auf einen Taxonomiepunkt angezeigt, welche Referenzen zu diesem Punkt existieren.

Ein weiteres Argument gegen das Vorliegen eines Elementarprozesses ist die Tatsache, dass dieser Elementarprozess weder eigenständig noch abgeschlossen ist. Er ist ganz offensichtlich integraler Bestandteil des Elementarprozesses *Suche über die Taxonomie*. Ohne die Anzeige der Taxonomie könnte der umfassende Elementarprozess nicht funktionieren.

Diese Seite unterstützt also nur einen Elementarprozess *Suche über die Taxonomie*. Es handelt sich bei diesem offensichtlich nicht primär um eine Eingabe, auch wenn der Benutzer einen Punkt der Taxonomie auswählt. Der Hauptzweck ist die Ausgabe von Daten. In Frage kommen somit zunächst Abfrage oder Ausgabe. Da die Verarbeitungsregel keine mathematische Formel oder Berechnung erwarten lässt, keine Daten abgeleitet werden, keine ILF gepflegt wird und sich auch das Systemverhalten nicht ändern wird, handelt es sich klar um eine Abfrage.

Online-Bibliographie: Suche nach Eigenschaften



The screenshot shows a Mozilla browser window titled "Bibliography Search - Mozilla". The address bar contains the URL "http://www.iste.uni-stuttgart.de/ps/reengineering/simple+search.html". The page content includes a navigation bar with "Home" and "Next: Regular Expressions". Below this is a section titled "Search for References" with a paragraph explaining the search process: "Fill out this form (you can leave out fields), press the button below, and you will receive all the references that contain the regular expressions you filled in. Note that the search is case-insensitive and that the entries to retrieve must have all specified attributes." The form consists of four dropdown menus labeled "author", "title", "keywords", and "abstract", each followed by a text input field. At the bottom of the form are two buttons: "Start searching" and "Clear all inputs". Below the buttons is a footer containing the email "koschke@informatik.uni-stuttgart.de (Feedback)", the copyright notice "Copyright © 1997 University of Stuttgart, Germany. \$Revision: 1.6 \$", and the last modified date "Last modified: Mon Jun 22 16:26:16 MET DST 1998".

Bibliography Search - Mozilla

File Edit View Go Bookmarks Tools Window Help

http://www.iste.uni-stuttgart.de/ps/reengineering/simple+search.html Search

Home Next: [Regular Expressions](#)

Search for References

Fill out this form (you can leave out fields), press the button below, and you will receive all the references that contain the [regular expressions](#) you filled in. Note that the search is case-insensitive and that the entries to retrieve must have **all** specified attributes.

author

title

keywords

abstract

koschke@informatik.uni-stuttgart.de (Feedback).

Copyright © 1997 University of Stuttgart, Germany. \$Revision: 1.6 \$

Last modified: Mon Jun 22 16:26:16 MET DST 1998

Auf dieser Seite können Referenzen anhand von Stichwörtern gesucht werden. Die Suche nach Stichwörtern kann auf verschiedene Felder der Referenzen eingeschränkt werden. Den Suchbereich kann man mit Hilfe einer Listbox für jedes Stichwort auswählen.

Hier tritt die interessante und häufig gestellte Frage auf, wie Listboxen zu behandeln sind. Können Sie auch einen Elementarprozess darstellen? Poensgen und Bock (2005) geben hierauf die Antwort, dass eben nicht Listboxen, sondern Elementarprozesse bewertet werden. Ob mit einer Listbox ein Elementarprozess zu werten ist, hängt vom Einzelfall ab. Die Listbox hier repräsentiert zulässige BibTeX-Felder. Damit werden keine Daten aus einem fachlichen Datenbestand dargestellt, sondern Metadaten, die ein korrektes Attribut im internen Datenbestand *Referenzen* beschreiben. Die Function-Point-Methode verlangt, dass bei einer Abfrage oder Ausgabe Daten oder Steuerinformationen über die Anwendungsgrenze verschickt werden. Hierbei sind explizit fachliche Daten gemeint. Dies ist jedoch bei den Elementen der Listbox nicht der Fall.

Ein andere Situation läge vor, wenn die Listboxen fachliche Daten repräsentieren würden, beispielsweise, wenn aus einer Liste verzeichneter Autoren ausgesucht werden könnte.

Wir halten also fest, dass wir die Auswahl in den Listboxen nicht als Elementarprozess betrachten. Nichtsdestotrotz stellen sie Felder dar, die Eingaben darstellen. Sie werden mit übertragen, damit der Wert im Suchfeld richtig interpretiert werden kann. Somit sind sie zumindest für die Zählung der DETs relevant.

Beispiel: Online-Bibliographie

Elementarprozesse und Datenbestände der Online-Bibliographie:

- El_1 : Neuen Artikel einpflegen
- EQ_1 : Beschreibung der Taxonomie
- EQ_2 : Suche über Taxonomie
- EQ_3 : Artikel über Suchmaske abfragen
- ILF_1 : Referenzen
- ILF_2 : Taxonomie-Metadaten

Function Points zusammenfassen

Unadjusted Function Points (UFP):

Parameter	Gewicht
EI	w_1
EO	w_2
EQ	w_3
ILF	w_4
ELF	w_5
UFP	$\sum w_i$

→ zu ermitteln: w_i

Umfang $\sim$ Summe FPs; „ungewichtete Funktionspunkte“ (UFP)

233 / 504

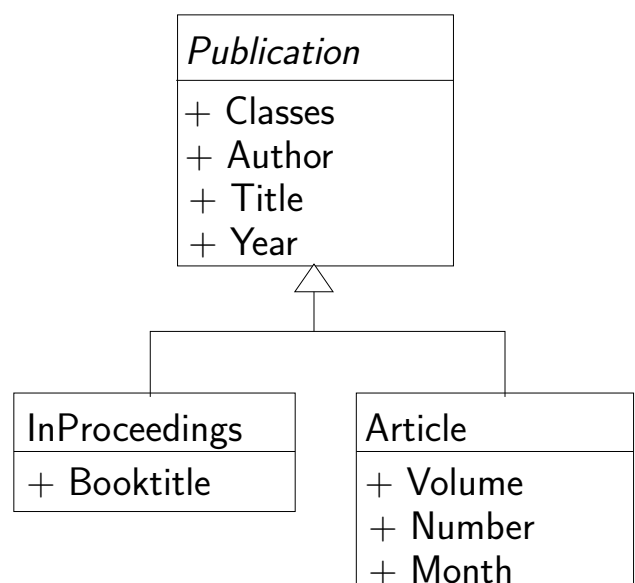
Beispiel: Komplexitätsgewichte w_i für ELF und ILF

Definition

Feldgruppe: RET (Record Element Type): für Benutzer erkennbare, logisch zusammengehörige Untergruppe von Datenelementen innerhalb eines Datenbestands (ILF, EIF)

Definition

Datenelementtypen: DET (Data Element Type): für Benutzer erkennbares, eindeutig bestimmtes, nicht-wiederholtes Feld



235 / 504

Hier werden die Daten abgeschätzt.

Der Datenbestand **Referenzen** unserer Online-Bibliographie-Beispiels wird hier beispielhaft (und fragmentarisch) durch ein Klassendiagramm beschrieben. Das angegebene Klassendiagramm hat insgesamt drei Klassen. Auf den ersten Blick scheinen hier drei Untergruppen zu existieren. Die Oberklasse ist jedoch abstrakt. Das heißt, dass eine solche Untergruppe gar nicht existiert bzw. nur eine logische Beschreibung darstellt, um Gemeinsamkeiten von anderen Untergruppen zusammen zu fassen. Abstrakte Klassen zählen somit nicht als Untergruppe. Die beiden anderen Klassen sind konkret. Somit ergeben sich zwei Feldgruppen: **RET** = 2. Die Datenelementtypen sind die Attribute aller Untergruppen einschließlich der von abstrakten Oberklassen ererbten. Davon gibt es acht: **DET** = 8. Die **Taxonomie-Metadaten** sind wie folgt strukturiert. Wir haben einerseits den Namen des Taxonomiepunkts, dann einen beschreibenden Text und schließlich einen Verweis auf die Oberklasse. Damit ergeben sich **RET** = 1 und **DET** = 3.

FP – Bestimmung der Komplexitätsgewichte w_i

Komplexitätsmatrizen:

- (Funktionstyp, #FTRs/RETs, #DETs) → FPs
- Zählen mittels Zählregeln pro Funktionstyp

		DETs		
FTRs/RETs	Funktionstyp	1 bis a	$a + 1$ bis b	$> b$
	1 bis x	gering	gering	mittel
	$x + 1$ bis y	gering	mittel	hoch
	$> y$	mittel	hoch	hoch

FTR: number of files updated or referenced. A record element type is a user recognizable subgroup of data elements within an ILF or EIF. A data element type is a unique user recognizable, nonrecursive, field.

Zählen von Datenelementtypen (DET)

Ein Datenelementtyp (DET) ist aus der Benutzersicht eindeutig bestimmbares, nicht rekursives Feld, das von der zu bewertenden externen Eingabe (EI) in einem internen Datenbestand (ILF) gepflegt wird.

Zählen Sie 1 DET für jedes aus Benutzersicht nicht rekursive Feld, das die Systemgrenze kreuzt und gebraucht wird, um den Elementarprozess abzuschließen.

Beispiel: Ein Texteingabefeld, in dem der Nachname eines neuen Kunden eingegeben wird, wird als 1 DET gezählt.

Gegenbeispiel: Eine Dateiauswahlliste, in der beliebig viele Dateien von der Festplatte des Benutzers ausgewählt werden können, ist rekursiv, und muss somit gesondert gezählt werden.

Zählen Sie keine Felder, die durch das System gesucht und/oder in einem ILF gespeichert werden, wenn die Daten nicht die Systemgrenze überqueren.

Zählen Sie nur 1 DET für die Fähigkeit, eine Systemantwort als Meldung für einen aufgetretenen Fehler aus der Anwendung heraus zu senden bzw. für die Bestätigung, dass die Verarbeitung beendet oder fortgesetzt werden kann.

Beispiel: Bei Eingabe eines Datums soll z.B. das Format TT/MM/JJJJ eingehalten werden. Gibt der Benutzer z.B. '12.03.1997' ein und bestätigt seine Eingabe, so erhält er die Meldung 'neuer Datensatz gespeichert'. Gibt er hingegen '12.3.97' ein (Jahreszahl nicht vierstellig) so erhält er die Fehlermeldung 'Fehler: Bitte Datum korrigieren'. Nur ein DET wird für diese Fähigkeit des Systems gezählt.

Zählen Sie nur 1 DET für die Möglichkeit, eine Aktion durchzuführen, auch wenn es viele Methoden gibt, die

denselben logischen Prozess anstoßen. Beispiel: In einem Eingabeformular gibt es einen OK-Button zum Absenden

der Daten. Die Tastaturkombination STRG-S führt ebenfalls zum Senden der Daten. Somit wird nur ein DET

gezählt.

(Fortsetzung)

Zählen von referenzierte Datenbeständen (FTR)

Ein referenzierter Datenbestand (FTR) ist eine vom Benutzer definierte Gruppierung zusammengehöriger Daten oder Steuerungsinformationen in einem internen Datenbestand (ILF), die bei der Bearbeitung der externen Eingabe gelesen oder gepflegt wird.

Zählen Sie 1 FTR für jeden referenzierten Datenbestand, der während der Verarbeitung der externen Eingabe gelesen wird. Beispiel: Es werden durch eine externe Eingabe Produktdaten in einer Datenbank gespeichert. Dazu werden die Produktbezeichnungen aus einer weiteren Datenbank ausgelesen, die damit zusätzlich zu der zu aktualisierenden Produktdatenbank einen weiteren Datenbestand darstellt, der jedoch nur gelesen wird.

Zählen Sie 1 FTR für jede ILF, die während der Verarbeitung der externen Eingabe gepflegt wird. Beispiel: Es wird zusätzlich zu den Aktionen des vorigen Beispiels eine Textdatei aktualisiert, in der die Anzahl der Zugriffe auf die Datenbank verzeichnet wird.

Zählen Sie nur 1 FTR für jede ILF, die während der Verarbeitung der externen Eingabe gelesen und gepflegt wird.

Beispiel: Würden die Informationen der Textdatei ebenfalls in der Datenbank gespeichert werden, so wird diese nur

als 1 FTR gezählt, obwohl die Datenbank zur Ein- und Ausgabe von Daten verwendet wird.

(Fortsetzung)

Besonderheiten bei grafischen Benutzungsoberflächen

Optionsfelder (Radiobuttons) stellen Datenelemente dar. Es wird pro Gruppe von Optionsfeldern 1 DET gezählt, da innerhalb einer Gruppe nur ein Optionsfeld ausgewählt werden kann. Beispiel: Eine Gruppe von 12 Radiobuttons, in der ein PKW-Typ ausgewählt werden kann, wird als 1 DET gezählt.

Kontrollkästchen (Checkboxes) stellen Datenelemente dar. Im Gegensatz zu Optionsfeldern können aus einer Gruppe von Checkboxes mehrere Elemente gleichzeitig ausgewählt werden. Somit wird jedes Element als 1 DET gezählt. Beispiel: Eine Gruppe von 12 Checkboxes, mit der ein Pizza-Belag zusammengestellt werden kann, wird als 12 DETs gezählt.

Eingabe- und Ausgabefelder stellen Datenelemente dar. Beispiel: In einer Bildschirmmaske werden Vorname, Nachname, Straße, Hausnummer, PLZ und Ort in Eingabefeldern erfasst. Somit werden 6 DET gezählt.

Literale stellen keine Datenelemente dar. Beispiel: Vor einem Feld ist die Textzeile 'monatliches Gehalt' und dahinter 'in Euro/Monat' angegeben. Beide Textzeilen sind Literale und werden nicht gezählt. Enter-, OK-Button und Programmfunktionstaste werden insgesamt als 1 DET gezählt, da jeweils die gleiche Funktion ausgeführt wird. Beispiel: Die Daten eines Dialogs werden nach Betätigen der Enter-Taste oder nach Betätigen der Schaltfläche 'übernehmen' (gleiche Funktion wie OK-Button) in einer Datenbank gespeichert. Es wird 1 DET gezählt.

Berichte/Reports können verschiedene Ausgabeformen haben. So kann die gleiche Datenbasis zur Darstellung als Tortendiagramm, Tabelle, textuell, als druckbares Format oder als Exportdatei dargestellt werden. Jedes Format stellt dabei eine externe Ausgabe (EO) dar.

FP – Werte der Komplexitätsgewichte w_i

		DET _s		
RET _s	ILF	1-19	20-50	51+
	1	7	7	10
	2-5	7	10	15
	6+	10	15	15

		DET _s		
RET _s	ELF	1-19	20-50	51+
	1	5	5	7
	2-5	5	7	10
	6+	7	10	10

Function Points zusammenfassen

Parameter	RET	DET	Gewicht
ILF ₁	2	8	7
ILF ₂	1	3	7
Parameter	FTR	DET	Gewicht
El ₁			
EQ ₁			
EQ ₂			
EQ ₃			
UFP			

239 / 504

Komplexitätsgewichte w_i für El, EO, EQ

Definition

Referenzierte Datenbestände: FTR (File Type Referenced):
von Elementarprozess verwendeter Datenbestand (ILF, ELF)

Beispiel: der Kundenstammdatenbestand, der bei der Ausgabe von Kundendaten herangezogen wird

Beispiele für DETs im Kontext von Funktionen:
Eingabe-/Ausgabefelder (GUI), Spalten u.Ä. bei Berichten

241 / 504

Online-Bibliographie: Neuen Artikel einpflegen

Submitting to the reengineering bibliography - Icedweasel

Home Prev: Submitting References

Submitting a reference to an article in a journal

Please fill out all of the mandatory fields. We appreciate if you fill in the optional fields as well. We are especially interested in the abstract and your classification according to the [taxonomy](#). For further questions we might also need your e-mail address.

[For an example on how to fill out a form, click here.](#)

After you have filled out the form click the "submit" button below.

Your e-mail address:

Mandatory fields

Please use "and" as an author separator as in: E. Chikofsky and James Cross.

Authors:

Title:

Journal:

Year:

Optional fields

Volume:

Number:

Pages:

Month:

242 / 504

Online-Bibliographie: Neuen Artikel einpflegen

Submitting to the reengineering bibliography - Icedweasel

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

If the contents of your reference is available via the web, please add the link information here:

URL:

Abstract:

Keywords:

Your personal note to this reference:

Classification:

The classification of the article is important to support retrieval by users interested in a certain topic. Therefore, we are very much interested in getting the article classified. And if you are the author there is no better expert to do it.

The classification list box below contains a hierarchy of bibliography-related terms. The indentation (stressed by dots) indicates the subclass relation. Choose any item you find appropriate by clicking on it. Clicking twice on the same item unselects it. Note that a parent class is automatically selected if you select one of its subclasses. Furthermore, you can select different classes for the same reference by multiple choices. Please be as specific as possible.

We tried to select terms that are as self-explanatory as possible. If something is not clear just see the [general description of the taxonomy](#).

Reengineering_in_General
- Reengineering_Collections
- Introductions_to_Reengineering
- Fundamentals

243 / 504

Beim Elementarprozesse *Neuen Artikel einpflegen* El_1 sind insgesamt 13 Textfelder auszufüllen. Außerdem muss am Ende der Schalter für das Absenden der Daten betätigt werden, um den Elementarprozess abzuschließen (ein Schalter für das Abbrechen zählt nicht, da damit der Prozess nicht abgeschlossen wird). Für diesen Elementarprozess ergeben sich also zunächst 14 DETs.

Außerdem soll der Artikel in der Taxonomie eingeordnet werden. Dies zählt als weitere Eingabemöglichkeit. Dabei können mehrere Klassen der Taxonomie ausgewählt werden. Der Datentyp der Auswahl, die dem System übergeben wird, stellt somit eine Liste von Taxonomieeinträgen dar. Die Taxonomieeinordnung zählt somit als ein 1 DET. Die Eingabe El_1 hat somit 15 DETs

Die Taxonomieeinordnung stellt keinen Elementarprozess dar, weil damit kein abgeschlossener Benutzerzweck verbunden ist. Sie ist nur sinnvoll im Kontext des Elementarprozesses *Neuen Artikel einpflegen*.

Wenn der Link *Classification* betätigt wird, wird die Taxonomie als Baum von Taxonomieeinträgen angezeigt. Dies zählt als ein DET (nicht die Anzahl von Daten sondern der Datentyp wird gezählt). Der Link *Classification* ist lediglich eine Navigation und zählt somit nicht. Die Abfrage EQ_1 hat somit nur einen DET.

Nun müssen noch die FTRs der beiden Elementarprozesse bestimmt werden. Bei der Eingabe eines neuen Artikels muss nur der Datenbestand *Referenzen* angefasst werden, bei der Abfrage der Taxonomie nur der Datenbestand *Taxonomie*.

Function Points zusammenfassen

- El_1 : Neuen Artikel einpflegen
- EQ_1 : Beschreibung der Taxonomie
- EQ_2 : Suche über Taxonomie
- EQ_3 : Artikel über Suchmaske abfragen
- ILF_1 : Referenzen
- ILF_2 : Taxonomie-Metadaten

Parameter	RET	DET	Gewicht
ILF_1	2	8	7
ILF_2	1	3	7
Parameter	FTR	DET	Gewicht
El_1	1	15	
EQ_1	1	1	
EQ_2			
EQ_3			
UFP			

FP – Werte der Komplexitätsgewichte w_i

		DET _s		
FTR _s	EI	1-4	5-15	16+
	0-1	3	3	4
	2	3	4	6
	3+	4	6	6

		DET _s		
FTR _s	EO	1-5	6-19	20+
	0-1	4	4	5
	2-3	4	5	7
	4+	5	7	7

		DET _s		
FTR _s	EQ	1-5	6-19	20+
	0-1	3	3	4
	2-3	3	4	6
	4+	4	6	6

245 / 504

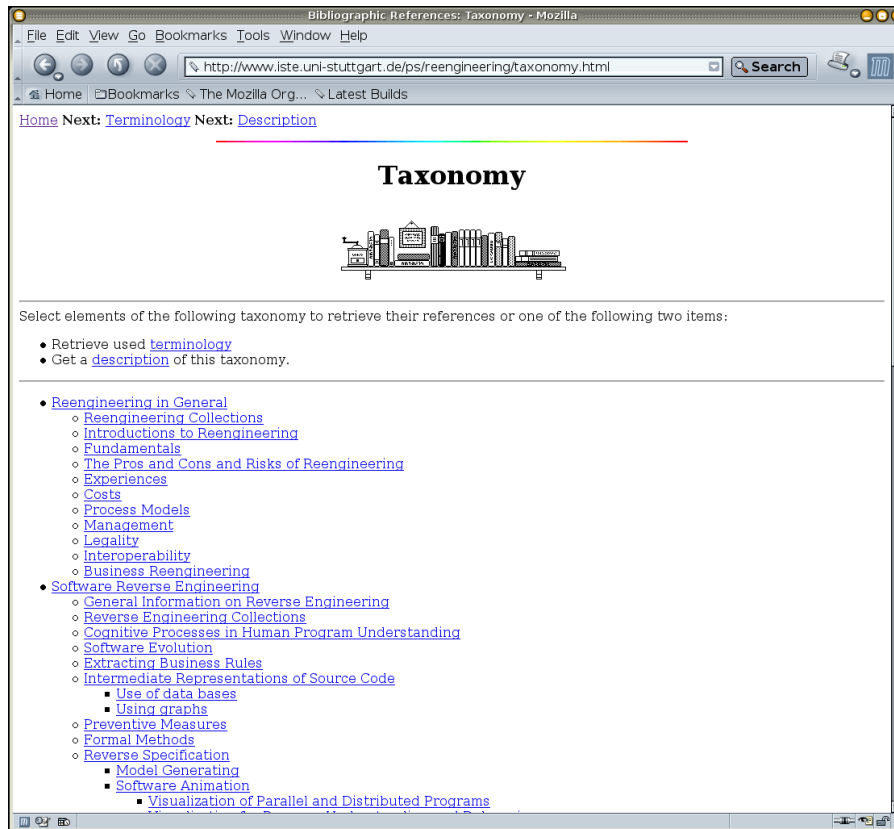
Function Points zusammenfassen

- EI₁: Neuen Artikel einpflegen
- EQ₁: Beschreibung der Taxonomie
- EQ₂: Suche über Taxonomie
- EQ₃: Artikel über Suchmaske abfragen
- ILF₁: Referenzen
- ILF₂: Taxonomie-Metadaten

Parameter	RET	DET	Gewicht
ILF ₁	2	8	7
ILF ₂	1	3	7
Parameter	FTR	DET	Gewicht
EI ₁	1	15	3
EQ ₁	1	1	3
EQ ₂			
EQ ₃			
UFP			

246 / 504

Online-Bibliographie: Taxonomiesuche



Annahme: Ein BibTeX-Eintrag habe max. 8 Einträge

247 / 504

Auf dieser Seite wird der Elementarprozess *Suche über die Taxonomie* erkennbar. Jeder Taxonomieeintrag stellt einen Link dar, der alle Artikel auflistet, die zu diesem Taxonomiepunkt gehören oder zu Taxonomiepunkten, die von diesem abgeleitet sind. Die Taxonomie selbst wird beim Start dieser Seite vom System bereit gestellt. Letzteres entspricht auf den ersten Blick dem Elementarprozess *Taxonomie anzeigen*. Wir müssen nun bestimmen, ob dieser Elementarprozess von dem weiter oben diskutierten unterscheidbar ist. Er unterscheidet sich nicht durch die Eingabe, die Verarbeitung oder den referenzierten Datenbeständen jedoch in den Ausgabedaten. Während beim ersten Elementarprozess *Beschreibung der Taxonomie* die einzelnen Taxonomiepunkte textuell beschrieben werden, wird uns auf dieser Seite beim Klicken auf einen Taxonomiepunkt angezeigt, welche Referenzen zu diesem Punkt existieren.

Ein weiteres Argument gegen das Vorliegen eines Elementarprozesses ist die Tatsache, dass dieser Elementarprozess weder eigenständig noch abgeschlossen ist. Er ist ganz offensichtlich integraler Bestandteil des Elementarprozesses *Suche über die Taxonomie*. Ohne die Anzeige der Taxonomie könnte der umfassende Elementarprozess nicht funktionieren.

Diese Seite unterstützt also nur einen Elementarprozess *Suche über die Taxonomie*. Es handelt sich bei diesem offensichtlich nicht primär um eine Eingabe, auch wenn der Benutzer einen Punkt der Taxonomie auswählt. Der Hauptzweck ist die Ausgabe von Daten. In Frage kommen somit zunächst Abfrage oder Ausgabe. Da die Verarbeitungsregel keine mathematische Formel oder Berechnung erwarten lässt, keine Daten abgeleitet werden, keine ILF gepflegt wird und sich auch das Systemverhalten nicht ändern wird, handelt es sich klar um eine Abfrage.

Zu beachten ist bei der Zählung der DETs generell, dass jedes DET nur einmal gezählt wird, auch wenn es in beide Richtungen der Systemgrenze übertragen wird.

Der Elementarprozess *Suche über die Taxonomie* EQ₂ ist mit einem DET für den ausgewählten Taxonomiepunkt assoziiert. Das Resultat ist eine Liste bibliographischer Referenzen, die der BibTeX-Struktur folgen. Da im Prinzip jede Art von Referenz zurückgeliefert werden kann, müssen wir die maximale Anzahl von Feldern aller BibTeX-Referenztypen bestimmen. Ohne in die Untiefen von BibTeX einzutauchen, nehmen wir der Einfachheit halber an, wir hätten maximal 8 Felder. Damit ergeben sich dann insgesamt 8+1=9 DETs für diesen Elementarprozess.

Der Elementprozess muss sowohl den Taxonomie- als auch den Referenzdatenbestand betrachten. Damit ergeben sich zwei FTRs.

Function Points zusammenfassen

- EI₁: Neuen Artikel einpflegen
- EQ₁: Beschreibung der Taxonomie
- EQ₂: Suche über Taxonomie
- EQ₃: Artikel über Suchmaske abfragen
- ILF₁: Referenzen
- ILF₂: Taxonomie-Metadaten

Parameter	RET	DET	Gewicht
ILF ₁	2	8	7
ILF ₂	1	3	7
Parameter	FTR	DET	Gewicht
EI ₁	1	15	3
EQ ₁	1	1	3
EQ ₂	2	9	4
EQ ₃			
UFP			

Online-Bibliographie: Suche nach Eigenschaften

The screenshot shows a Mozilla browser window titled "Bibliography Search - Mozilla". The address bar contains the URL "http://www.iste.uni-stuttgart.de/ps/reengineering/simple+search.html". The browser's menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, and Help. Below the menu bar, there are navigation buttons (Back, Forward, Home, Stop) and a search button. The main content area has a link "Home Next: Regular Expressions" and a section titled "Search for References". Below this title, there is a paragraph explaining the search process: "Fill out this form (you can leave out fields), press the button below, and you will receive all the references that contain the [regular expressions](#) you filled in. Note that the search is case-insensitive and that the entries to retrieve must have **all** specified attributes." The form consists of four dropdown menus labeled "author", "title", "keywords", and "abstract", each followed by a text input field. At the bottom of the form, there are two buttons: "Start searching" and "Clear all inputs". Below the buttons, there is a link "koschke@informatik.uni-stuttgart.de (Feedback)", a copyright notice "Copyright © 1997 University of Stuttgart, Germany. \$Revision: 1.6 \$", and a modification date "Last modified: Mon Jun 22 16:26:16 MET DST 1998". The browser's status bar at the bottom shows "Done".

249 / 504

Auf dieser Seite können Referenzen anhand von Stichwörtern gesucht werden. Die Suche nach Stichwörtern kann auf verschiedene Felder der Referenzen eingeschränkt werden. Den Suchbereich kann man mit Hilfe einer Listbox für jedes Stichwort auswählen.

Hier tritt die interessante und häufig gestellte Frage auf, wie Listboxen zu behandeln sind. Können Sie auch einen Elementarprozess darstellen? Poensgen und Bock (2005) geben hierauf die Antwort, dass eben nicht Listboxen, sondern Elementarprozesse bewertet werden. Ob mit einer Listbox ein Elementarprozess zu werten ist, hängt vom Einzelfall ab. Die Listbox hier repräsentiert zulässige BibTeX-Felder. Damit werden keine Daten aus einem fachlichen Datenbestand dargestellt, sondern Metadaten, die ein korrektes Attribut im internen Datenbestand *Referenzen* beschreiben. Die Function-Point-Methode verlangt, dass bei einer Abfrage oder Ausgabe Daten oder Steuerinformationen über die Anwendungsgrenze verschickt werden. Hierbei sind explizit fachliche Daten gemeint. Dies ist jedoch bei den Elementen der Listbox nicht der Fall.

Ein andere Situation läge vor, wenn die Listboxen fachliche Daten repräsentieren würden, beispielsweise, wenn aus einer Liste verzeichneter Autoren ausgesucht werden könnte.

Wir halten also fest, dass wir die Auswahl in den Listboxen nicht als Elementarprozess betrachten. Nichtsdestotrotz stellen sie Felder dar, die Eingaben darstellen. Sie werden mit übertragen, damit der Wert im Suchfeld richtig interpretiert werden kann. Somit sind sie zumindest für die Zählung der DETs relevant.

Insgesamt ergeben sich seitens der Eingabe durch den Benutzer für den Elementarprozess EQ₃ vier DETs für die Textfelder, weitere vier DETs für die Listboxen und schließlich noch ein DET für den Schalter, um die Suche zu starten, der den Elementarprozess anstößt.

Als Resultat erhalten wir eine Liste aller Referenzen, die die angegebenen Suchkriterien erfüllen. Da auch hier wieder die Referenztypen sehr unterschiedlich sein können, gehen wir wie bereits weiter oben von einer Obergrenze von 8 verschiedenen BibTeX-Attributen aus.

Die Gesamtzahl der DETs für die Eingabe und Ausgabe dieses Elementarprozesses beträgt somit 9+8=17.

Die Suche ist unabhängig von der Taxonomie, so dass nur der Datenbestand *Referenzen* angefasst werden muss. Es ergibt sich damit ein FTR.

Function Points zusammenfassen

- EI₁: Neuen Artikel einpflegen
- EQ₁: Beschreibung der Taxonomie
- EQ₂: Suche über Taxonomie
- EQ₃: Artikel über Suchmaske abfragen
- ILF₁: Referenzen
- ILF₂: Taxonomie-Metadaten

Parameter	RET	DET	Gewicht
ILF ₁	2	8	7
ILF ₂	1	3	7
Parameter	FTR	DET	Gewicht
EI ₁	1	15	3
EQ ₁	1	1	3
EQ ₂	2	9	4
EQ ₃	1	17	4
UFP			28

FP – Gewichtete Function-Points

Systemmerkmale:

- Datenkommunikation
- Verteilte Verarbeitung
- Leistungsanforderungen
- Ressourcennutzung
- Transaktionsrate
- Online-Benutzerschnittstelle
- Benutzerfreundlichkeit
- Online-Verarbeitung
- Komplexe Verarbeitung
- Wiederverwendbarkeit
- Migrations-/Installationshilfen
- Betriebshilfen
- Mehrfachinstallationen
- Änderungsfreundlichkeit

Bewertung: 0 = kein Einfluss, 5 = starker Einfluss

252 / 504

Konkrete Fragen I

- Are data communications required? **1**
- Are there distributed processing functions? **0**
- Is performance critical? **1**
- Will the system run in an existing, heavily utilized operational environment? **1**
- How frequently are transactions executed? **3**
- Does the system require on-line data entry? **4**
- Was the application designed for end-user efficiency? **0**

253 / 504

Konkrete Fragen II

- Are the master files updated on-line? **0**
- Is the internal processing complex? **1**
- Is the code designed to be reusable? **1**
- Are conversion and installation included in the design? **0**
- How effective and/or automated are start-up, back-up, and recovery procedures? **0**
- Is the system designed for multiple installations in different organizations? **2**
- Is the application designed to facilitate change and ease of use by the user? **0**

FP – Gewichtete Function-Points

- TDI (Total Degree of Influence) = Summe der Bewertungen
 $\in [0 \dots 14 * 5 = 70]$
- VAF (Value Adjustment Factor) = $TDI/100 + 0,65$
→ Gesamteinflussfaktor: 65% - 135%
- AFP (Adjusted Function Points) = $UFP \cdot VAF$

Beispiel:

- $TDI = 14$
- $VAF = 14/100 + 0,65 = 0,79$
- $AFP = 28 \cdot 0,79 = 23$ (es wird stets aufgerundet)

255 / 504

Kritik an der Function-Point-Methode

Einwand gegen Systemmerkmale:

- einige der Systemmerkmale sind heute eher anachronistisch
 - durch Multiplikation von VAF und UAF findet eine fragwürdige Vermengung von technischen Faktoren und funktionalem Umfang sowie von quantitativen und qualitativen Aspekten statt
 - VAF bringt kaum praktischen Nutzen:
 - COCOMO verwendet UAF als Eingangsgröße
 - COCOMO hat eigene technische und organisatorische Gewichtungsfaktoren
- AFP sind heute eher unbedeutend (im Gegensatz zu UFP)
- bei Angaben von Function-Points nachfragen: AFP oder UFP?

256 / 504

Kritik an der Function-Point-Methode

Allgemeinere Einwände:

- sehr stark auf Informationssysteme bezogen; Übertragbarkeit auf andere Arten, wie z.B. reaktive Systeme oder Compiler, fragwürdig
- die Komplexität der Anfragen kann maximal um den Faktor 2 variieren; die Komplexität der Verarbeitungslogik geht nicht direkt ein

257 / 504

FP – Umrechnung in Aufwand

Gesucht: Abbildung FPs → Aufwand

- Erstellung einer neuen Erfahrungskurve
(Zählen abgeschlossener Projekte, Regressionsanalyse)
- Datenbank, z.B. ISBSG³:
 - 3GL-Projekte: $PM = 0,971 \cdot AFP^{0,351}$
 - 4GL-Projekte: $PM = 0,622 \cdot AFP^{0,405}$
 - basierend auf 662 Projekten: $PM = 0,38 \cdot AFP^{0,37}$
- grobe Schätzung mit Faustregeln Jones (1996):
 - Entwicklungsdauer (Monate) = $AFP^{0,4}$
 - Personen = $AFP / 150$ (aufgerundet)
 - Aufwand (Personenmonate) = Personen · Entwicklungsdauer

Beispiel (mit Jones-Schätzung):

- Entwicklungsdauer (Monate) = $23^{0,4} = 3,5$
- Personen = $23/150 \rightarrow 1$
- Aufwand (Personenmonate) = $1 \cdot 3,5 = 3,5$

³International Software Benchmarking Standards Group

FP – Umrechnung in LOC

Mittlere Anzahl Codezeilen pro FP (Jones 1995):

Sprache	ØLOC
Assembler	320
C	128
FORTRAN	107
COBOL (ANSI 85)	91
Pascal	91
C++	53
Java	53
Ada 95	49
Smalltalk	21
SQL	12

Bewertung der Function-Point-Methode

- + Wird als beste verfügbare Methode zur Schätzung kommerzieller Anwendungssysteme angesehen (Balzert 1997)
- + Sinnvoll einsetzbar, wenn Erfahrungswerte von vergleichbaren Projekten vorliegen (Kemerer 1987)
- Bewertung der Systemmerkmale subjektiv (Symons 1988)
- + Studie: mittlere FP-Abweichung zwischen zwei „Zählern“ nur 12% (Kemerer und Porter 1992)
- Zählen der FPs relativ aufwendig

260 / 504

Object Points

- Für 4GLs (Query Languages, Report Writers, ...)
- Haben nicht unbedingt mit OOP-Objekten zu tun
- Gewichtete Schätzung von *Objects Points* =
 - Anzahl verschiedener „Screens“
 - Anzahl erstellter „Reports“
 - Anzahl zu entwickelnder 3GL-Module
- Vorteil: Einfacher und weniger zu schätzen:
vergleichbare Präzision wie Function-Point-Schätzung (Banker u. a. 1991)
47% des Aufwands für Function-Point-Schätzung (Kauffman und Kumar 1993)

261 / 504

Object Points

Screens

# views contained	# data tables		
	< 4	< 8	8+
< 3	1	1	2
3-7	1	2	3
≥ 8	2	3	3

Reports

# sections contained	# data tables		
	< 4	< 8	8+
0-1	2	2	5
2-3	2	5	8
4+	5	8	8

Views: Menge logisch zusammengehöriger Daten (z.B. Kundenstammdaten)

Data Tables = # Server data tables + # Client data tables
(Tabellen, die abgefragt werden müssen, um Daten zu bestimmen)

Jede 3GL-Komponente: 10 object points

262 / 504

COCOMO

COCOMO = Constructive Cost Model (Boehm 1981)

- Eingaben: Systemgröße, Projektrahmenbedingungen
- Ausgaben: Realisierungsaufwand, Entwicklungszeit
- Basiert auf Auswertung sehr vieler Projekte

263 / 504

COCOMO II

Unterscheidung nach Phasen (Boehm u. a. 2000):

- Frühe Prototypenstufe
- Frühe Entwurfsstufe
- Stufe nach Architekturentwurf

Spätere Schätzung → höhere Genauigkeit

264 / 504

COCOMO II – Early prototyping level

Eingaben:

- Object Points (OP)
- Produktivität (PROD):

Erfahrung/Fähigkeiten der Entwickler	-	-	-	o	+	++
Reife/Fähigkeiten der CASE-Tools	-	-	-	o	+	++
Median obiger Zeilen	-	-	-	o	+	++
PROD [NOP/Monat]	4	7	13	25	50	

- Wiederverwendungsanteil *%reuse* in Prozent

Abgeleitete Größen:

- New Object Points (NOP): berücksichtigen Wiederverwendung
$$NOP = OP \cdot (100 - \%reuse)/100$$
- Aufwand in Personenmonaten $PM = NOP/PROD$

265 / 504

COCOMO II – Early design level

- Eingabe: FP → LOC
- Ausgabe: Personenmonate
 $PM_{NS} = A \cdot KLOC^E \cdot EM + PM_m$ bei nominalem Zeitplan

$$E = B + \sum w_i / 100$$

w_i : 5 = sehr klein, 0 = sehr groß:

- Erfahrung mit Domäne
- Flexibilität des Entwicklungsprozesses
- Risikomanagement
- Teamzusammenhalt
- Prozessreife

Schätzung basiert auf Function Points (LOCs werden daraus abgeleitet). $A = 2,94$ in initialer Kalibrierung (empirischer Wert) Nach Kalibrierung für COCOMO II.2000 $B = 0,91$; ein Wert > 1 wäre plausibler.

COCOMO II – Early design level

$$PM_{NS} = A \cdot KLOC^E \cdot EM + PM_m$$

$$EM = \prod Effort-Multiplier_i$$

7 lineare Einflussfaktoren (6 Stufen, Standard: 1,00, in Tabelle nachschlagen)

- Produktgüte
- Produktkomplexität
- Plattformkomplexität
- Fähigkeiten des Personals
- Erfahrung des Personals
- Zeitplan
- Infrastruktur

COCOMO II – Early design level

$$PM_{NS} = A \cdot \text{KLOC}^E \cdot EM + PM_m$$

Korrekturfaktor PM_m bei viel generiertem Code

höhere Produktivität bei Code-Generierung; nicht weiter diskutiert hier.

Faktoren für Exponent E I

$$PM_{NS} = A \cdot KLOC^E \cdot EM + PM_m$$

- Erfahrung mit Anwendungsbereich (PREC)
 - Erfahrung mit vorliegendem Projekttyp
 - 5 keine Erfahrung
 - 0 vollständige Vertrautheit
- Entwicklungsflexibilität (FLEX)
 - Grad der Flexibilität im Entwicklungsprozess
 - 5 Prozess vom Kunden fest vorgegeben
 - 0 Kunde legt nur Ziele fest
- Risikomanagement (RESL)
 - Umfang der durchgeführten Risikoanalyse
 - 5 keine Risikoanalyse
 - 0 vollständige und genaue Risikoanalyse

269 / 504

Faktoren für Exponent E II

- Teamzusammenhalt (TEAM)
 - Vertrautheit und Eingespieltheit des Entwicklungsteams
 - 5 schwierige Interaktionen
 - 0 integriertes und effektives Team ohne Kommunikationsprobleme
- Prozessreife (EPML)
 - Reife des Entwicklungsprozesses (z.B. CMM);
 - beabsichtigt: gewichteter Anteil der mit „ja“ beantworteten Fragen im CMM-Fragebogen
 - pragmatisch: CMM-Level
 - 5 niedrigster CMM-Level
 - 0 höchster CMM-Level

270 / 504

Effort Multiplier RCPX: Product Reliability and Complexity

$$PM_{NS} = A \cdot KLOC^E \cdot EM + PM_m \text{ mit } EM = \prod Effort-Multiplier_i$$

RELY Required reliability
 DOCU Documentation match to life-cycle needs
 CPLX Product complexity
 DATA Data base size

Grad: Punkte:	very low 1	low 2	nominal 3	high 4	very high 5	extra high 6	
RCPX							
RELY	very little	little	some	basic	strong		
DOCU	very little	little	some	basic	strong		
CPLX	very little	little	some	basic	strong		
DATA		small	moderate	large	very large		
$\sum$ Punkte:	5–6	7–8	9–11	12	13–15	16–18	19–21
EM_{RPCX}	0,49	0,60	0,83	1,00	1,33	1,91	2,72

271 / 504

DATA: This cost driver attempts to capture the effect large test data requirements have on product development. The rating is determined by calculating D/P, the ratio of bytes in the testing database to SLOC in the program. The reason the size of the database is important to consider is because of the effort required to generate the test data that will be used to exercise the program. In other words, DATA is capturing the effort needed to assemble and maintain the data required to complete test of the program.

Effort Multiplier PDIF: Platform Difficulty

TIME Execution time constraints (Auslastung CPU)
 STOR Main storage constraints (Auslastung RAM)
 PVOL Platform volatility (Häufigkeit von Plattformänderungen)

Grad: Punkte:	low 2	nominal 3	high 4	very high 5	extra high 6
PDIF					
TIME		≤50%	≤65%	≤80%	≤90%
STORE		≤50%	≤65%	≤80%	≤90%
PVOL	very stable	stable	somewhat volatile	volatile	
$\sum$ Punkte:	8	9	10–12	13–15	16–17
EM_{PDIF}	0,87	1,00	1,29	1,81	2,61

272 / 504

Effort Multiplier PERS: Personnel Capability

ACAP Analyst capability (gemessen als Perzentil)
 PCAP Programmer capability (gemessen als Perzentil)
 PCON Personnel continuity (gemessen durch Personalfluktuation)

Grad: Punkte:	very low 1	low 2	nominal 3	high 4	very high 5		
PERS							
ACAP	15%	35%	55%	75%	90%		
PCAP	15%	35%	55%	75%	90%		
PCON	48%	24%	12%	6%	3%		
$\sum$ Punkte:	3–4	5–6	7–8	9	10–11	12–13	14–15
EM_{PERS}	2,12	1,62	1,26	1,00	0,83	0,63	0,50

273 / 504

A percentile rank is the proportion of scores in a distribution that a specific score is greater than or equal to. For instance, if you received a score of 95 on a math test and this score was greater than or equal to the scores of 88% of the students taking the test, then your percentile rank would be 88. You would be in the 88th percentile.

Effort Multiplier PREX: Personnel Experience

AEXP Applications experience
 PLEX Platform experience
 LTEX Language/tool experience

Grad: Punkte:	very low 1	low 2	nominal 3	high 4	very high 5		
PREX							
AEXP	≤2 Mo.	6 Mo.	1 J.	3 J.	6 J.		
PLEX	≤2 Mo.	6 Mo.	1 J.	3 J.	6 J.		
LTEX	≤2 Mo.	6 Mo.	1 J.	3 J.	6 J.		
$\sum$ Punkte:	3–4	5–6	7–8	9	10–11	12–13	14–15
EM_{PREX}	1,59	1,33	1,22	1,00	0,87	0,74	0,62

Effort Multiplier FCIL: Facilities

TOOL	Use of software tools						
SITE	Multisite development						
Grad:	very low	low	nominal	high	very high		
Punkte:	1	2	3	4	5	6	
FCIL							
TOOL	(1)	(2)	(3)	(4)	(5)		→ nächste Folie
SITE	(1)	(2)	(3)	(4)	(5)	(6)	→ nächste Folie
$\sum$ Punkte:	2	3	4–5	6	7–8	9–10	11
EM_{FCIL}	1,43	1,30	1,10	1,00	0,87	0,73	0,62

275 / 504

TOOL und SITE

TOOL:

- ① Editor, Compiler, Debugger
- ② einfaches CASE-Werkzeug, schlechte Integration
- ③ Basis-Life-Cycle-Tools moderat integriert
- ④ weitergehende, reife Life-Cycle-Tools moderat integriert
- ⑤ weitergehende, proaktive Life-Cycle-Tools gut integriert mit Prozessen, Methoden und Wiederverwendung

SITE:

- ① Telefon prinzipiell vorhanden und Post
- ② individuelles Telefon und Fax
- ③ E-Mail (niedrige Bandbreite)
- ④ elektronische Kommunikation mit großer Bandbreite
- ⑤ elektronische Kommunikation mit großer Bandbreite, gelegentliche Videokonferenzen
- ⑥ interaktive Multimedia

276 / 504

Effort Multiplier SCED: Schedule

Es besteht Notwendigkeit, den Zeitplan zu straffen bzw. es wird mehr Zeit als notwendig eingeräumt.

SCED = Verkürzung bzw. Verlängerung des nominalen Zeitplans.

	75%	85%	100%	130%	160%
EM_{SCED}	1,43	1,14	1,00	1,00	1,00

277 / 504

This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. Accelerated schedules tend to produce more effort in the later phases of development because more issues are left to be determined due to lack of time to resolve them earlier. A schedule compress of 74% is rated very low. A stretch-out of a schedule produces more effort in the earlier phases of development where there is more time for thorough planning, specification and validation. A stretch-out of 160% is rated very high.

Stretch-outs (i.e., SCED > 100) do not add or decrease effort. Their savings because of smaller team size are generally balanced by the need to carry project administrative functions over a longer period of time.

Rechenbeispiel

Nominaler Aufwand

Personenmonate $PM_{NS} = A \cdot \text{KLOC}^E \cdot EM$ mit $EM_{SCED} = 1,0$

mit $A = 2,94$ und $E = B + \sum w_i/100$ mit $B = 0,91$.

Annahme: es herrschen einfache Verhältnisse:

- $\forall i : w_i = 0 \Rightarrow E = 0,91$ (bester Fall)
- nominale Effort-Multiplier = 1,00 (Normalfall) $\Rightarrow EM = 1,00$

Geschätzte Programmlänge = 100 [KLOC]

→ $PM_{NS} = 2,94 \times 100^{0,91} \times 1,0 = 194,24$ Monate ≈ 16 Jahre

278 / 504

Entwicklungsdauer

Nominale Entwicklungsdauer (Kalenderzeit in Monaten)

$$TDEV_{NS} = C \times PM_{NS}^{D+0,2 \times (E-B)}$$

mit $C = 3,67$ und $D = 0,28$.

Beispiel: $TDEV_{NS} = 3,67 \times 194,24^{0,28+0,2 \times (0,91-0,91)} \approx 16$

Anzahl Entwickler

$$N = PM_{NS} / TDEV_{NS}$$

Beispiel: $N = 194,24 / 16 = 12$

279 / 504

Verkürzte Entwicklungsdauer

Chef: „Wieso 16 Monate? Geht das nicht schneller?“

PM_{NS} geht von $SCED = 1,0$ aus.

Abweichung von der nominalen Entwicklungsdauer

$$TDEV = TDEV_{NS} \times SCED$$

Wir verkürzen auf 75%:

$$TDEV = 16 \times 75/100 = 12 \text{ Monate}$$

Chef: „Super!“

Wir setzen $SCED = 75$ in PM-Formel ein.

$$PM = 2,94 \times 100^{0,91} \times 1,0 \times 1,43 = 277,76$$

Erhöhung des Aufwands: um 43 %

Chef: „43% mehr Kosten? Seid Ihr wahnsinnig?“

280 / 504

COCOMO II – Post-architecture level

Berücksichtigt:

- Auswirkungen erwarteter Änderungen von Anforderungen
- Ausmaß/Aufwand der möglichen Wiederverwendung
 - Aufwand für Entscheidung, ob Wiederverwendung
 - Aufwand für das Verstehen existierenden Codes
 - Aufwand für Anpassungen
- 17 verfeinerte lineare Einflussfaktoren
- Schätzung basiert auf LOC

281 / 504

COCOMO II – Post-architecture level

Produktgüte/-kompl	→	Verlässlichkeit, Datenbasisgröße, Komplexität, Dokumentation
Plattformkomplexität	→	Laufzeit-, Speicherbeschränkungen, Plattformdynamik
Fähigkeiten Personal	→	Fähigkeiten der Analysten/Entwickler, Kontinuität des Personals
Erfahrung Personal	→	Domänenenerfahrung der Analysten/Entwickler, Erfahrung mit Sprache und Werkzeugen
Infrastruktur	→	Tools, verteilte Entwicklung+Kommunikation

282 / 504

Einflussfaktoren (Cost Drivers) für Cocomo-2

	- -	-	o	+	++	+++
Product Attributes						
RELY – Required reliability	0,82	0,92	1,00	1,10	1,26	
DATA – Data base size		0,90	1,00	1,14	1,28	
CPLX – Product Complexity	0,73	0,87	1,00	1,17	1,34	1,74
RUSE – Required Reusability		0,95	1,00	1,07	1,15	1,24
DOCU – Doc, match to life-cycle needs	0,81	0,91	1,00	1,11	1,23	
Computer Attributes						
TIME – Execution time constr.			1,00	1,11	1,29	1,63
STOR – Main storage constr.			1,00	1,05	1,17	1,46
PVOL – Platform volatility		0,87	1,00	1,15	1,30	

283 / 504

Einflussfaktoren (Cost Drivers) für Cocomo-2

	- -	-	o	+	++	+++
Personell attributes						
ACAP – Analyst capability	1,42	1,19	1,00	0,85	0,71	
PCAP – Programmer capability	1,34	1,15	1,00	0,88	0,76	
AEXP – Applications experience	1,22	1,10	1,00	0,88	0,81	
PLEX – Platform experience	1,19	1,09	1,00	0,91	0,85	
LTEX – Language/tool exp.	1,20	1,09	1,00	0,91	0,84	
Project attributes						
TOOL – Use of software tools	1,17	1,09	1,00	0,90	0,78	
SITE – Multisite development	1,22	1,09	1,00	0,93	0,86	0,80
SCED – Required dev. schedule	1,43	1,14	1,00	1,00	1,00	

284 / 504

Zusammenfassung

- alle Schätzungen basieren auf Erfahrung
- kontinuierlich schätzen
- verschiedene Techniken anwenden

Weiterführende Literatur

- Poensgen und Bock (2005) fassen auf 160 Seiten das Wichtigste zur Function-Point-Analyse zusammen. Hilfreich für das Verständnis ist ein umfangreiches Beispiel, bei dem die Unadjusted Function Points für ein Microsoft-Adressbuch ermittelt werden.
- Das Buch von Boehm u. a. (2000) ist die Referenz für COCOMO II; das Buch ist sehr umfassend und detailliert; es beschreibt Praxisbeispiele und die Kalibrierung des Modells mit neuen Daten.

286 / 504

Wiederholungs- und Vertiefungsfragen I

- Welche Möglichkeiten zur Schätzung von Aufwand und Kosten für die Software-Entwicklung gibt es?
- Wann wird geschätzt?
- Erläutern Sie die Function-Point-Methode (am konkreten Beispiel).
- Was sind Adjusted Function-Points im Unterschied zu Unadjusted-Function-Points?
- Wie errechnet sich der Aufwand aus den Function-Points?
- Was ist die Idee von Object-Points im Gegensatz zu Function-Points?
- Erläutern Sie die CoCoMo-Methode (neue Version) für die Level *Early Prototyping* und *Early Design Level*.
- Geben Sie die grundsätzliche Formel für den Aufwand wieder. Was bedeuten die verschiedenen Parameter?

287 / 504

Wiederholungs- und Vertiefungsfragen II

- Um welche Art von Parametern handelt es sich bei den Faktoren, die im Exponenten auftreten?
- Wozu die Unterscheidung in die verschiedene Stufen (Level)?
- Wie errechnet sich die Entwicklungsdauer aus dem Aufwand?
- Wie wird eine Verkürzung der nominalen Entwicklungsdauer behandelt?
- Vergleichen Sie die Function-Point-Methode mit CoCoMo.

N.B.:

- 1 Die Übungsaufgaben sind weitere Beispiele relevanter Fragen.
- 2 Bei der Darstellung der Einflussfaktoren für die Adjusted Function Points genügt es, ein paar Beispiele nennen zu können und zu wissen, worauf sie sich grundsätzlich beziehen (System und nicht Entwicklungsteam/-prozess); keinesfalls wird Gedächtnisleistung abgeprüft; das gilt auch für die Einflussfaktoren von CoCoMo.

288 / 504

Wiederholungs- und Vertiefungsfragen III

- 3 Bei der Darstellung von Function-Points und CoCoMo interessieren nicht die absoluten Zahlen, aber sehr wohl der grundsätzliche Aufbau der Formeln.

289 / 504

Entwurfsmuster

Entwurfsmuster

Kategorien von Entwurfsmustern
Bestandteile eines Entwurfsmusters
Entwurfsmuster Singleton
Entwurfsmuster Adapter
Entwurfsmuster Command
Entwurfsmuster Decorator
Entwurfsmuster Strategy
Entwurfsmuster für Synchronisation: Scoped Locking
Entwurfsmuster für Synchronisation: Strategized Locking
Entwurfsmuster für Synchronisation: Thread-Safe Interface
Entwurfsmuster für Parallelisierung: Thread Pool
Entwurfsmuster für Parallelisierung: Monitor Object
Entwurfsmuster für Parallelisierung: Leader/Followers
Wiederholungsfragen

290 / 504

Lernziele

- Verstehen, was Entwurfsmuster sind
- Verschiedene Entwurfsmuster kennen und anwenden können
- Qualitäten und Einsetzbarkeit der Entwurfsmuster kennen

291 / 504

Entwurfsmuster

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Christopher Alexander (Architekt und Mathematiker),
“A pattern language”, 1977.

Definition

Entwurfsmuster: “Musterlösung” für ein wiederkehrendes Entwurfsproblem.

293 / 504

Kategorien von Entwurfsmustern

- Muster für das Erzeugen von Instanzen
 - Singleton
- strukturelle Muster zur Komposition von Klassen oder Objekten
 - Composite
 - Adapter
 - Decorator
- Verhaltensmuster betreffen Interaktion von Klassen oder Objekten
 - Command

294 / 504

Bestandteile eines Entwurfsmusters

- **Name** (kurz und beschreibend)
- **Problem**: Was das Entwurfsmuster löst
- **Lösung**: Wie es das Problem löst
- **Konsequenzen**: Folgen und Kompromisse des Musters.

295 / 504

Problem

- es soll nur eine einzige Instanz einer Klasse geben, die global verfügbar sein soll
- Beispiele:
 - Zentrales Protokoll-Objekt, das Ausgaben in eine Datei schreibt.
 - Druckaufträge, die zu einem Drucker gesendet werden, sollten nur in einen einzigen Puffer geschrieben werden.

296 / 504

Entwurfsmuster Singleton (Einzelstück)

Lösung (Muster für das Erzeugen von Instanzen):

Singleton
- static instance: Singleton - static ...
+ static Singleton getInstance() - Singleton() + ...

297 / 504

Code

```
1 public final class Singleton {
2
3     private static Singleton instance;
4     // speichert einzige Instanz
5
6     private Singleton() {}
7     // kann von außerhalb nicht benutzt werden
8
9     // liefert einzige Instanz
10    public synchronized static Singleton getInstance() {
11        if (instance == null) { // lazy instantiation
12            instance = new Singleton();
13        }
14        return instance;
15    }
16 }
```

298 / 504

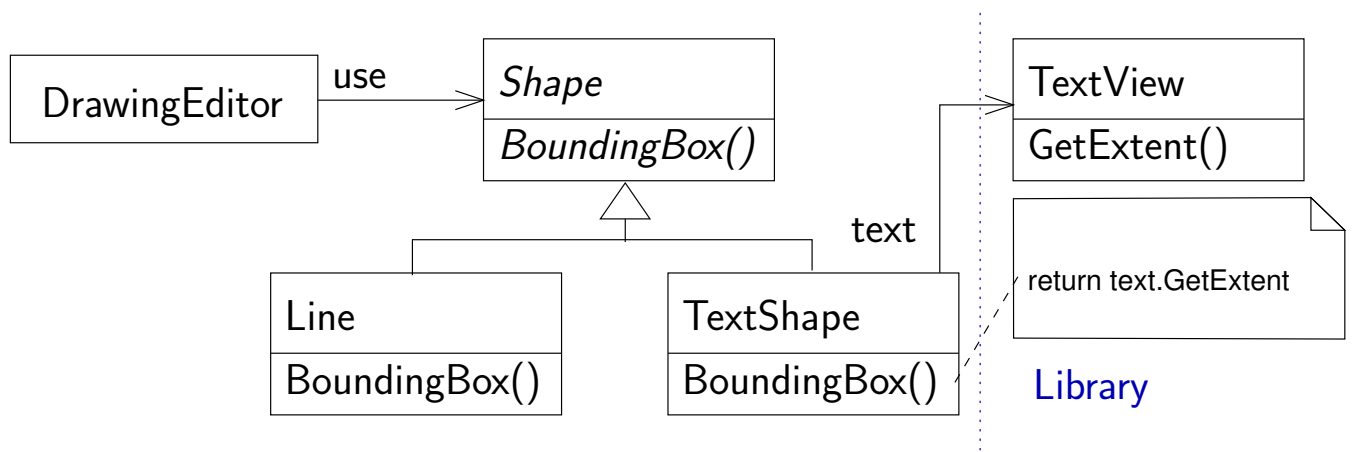
Konsequenzen

- Singleton hat strikte Kontrolle über wie und wann Klienten es verwenden
- vermeidet globale Variablen
- leicht erweiterbar, um mehrere Instanzen zuzulassen
- Variante: Singleton kann spezialisiert werden

299 / 504

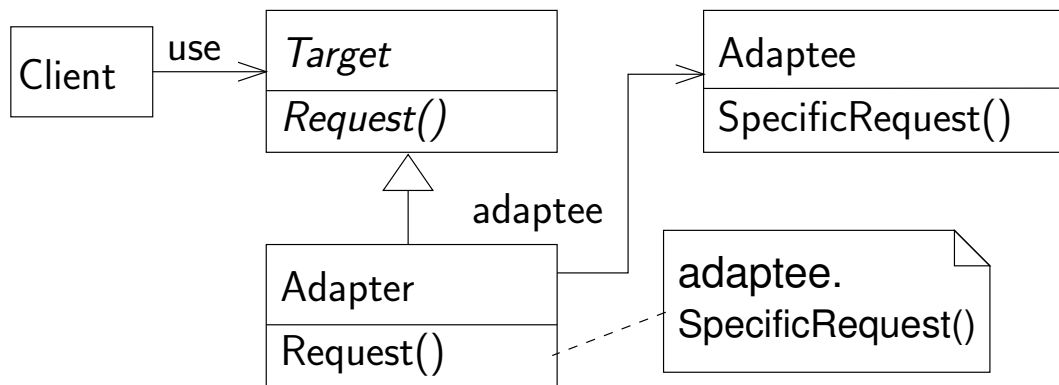
Problem

- eine vorhandene wiederverwendbare Komponente hat nicht die passende Schnittstelle,
- und der Code der Komponente kann nicht verändert werden



300 / 504

Lösung: Objekt-Adapter

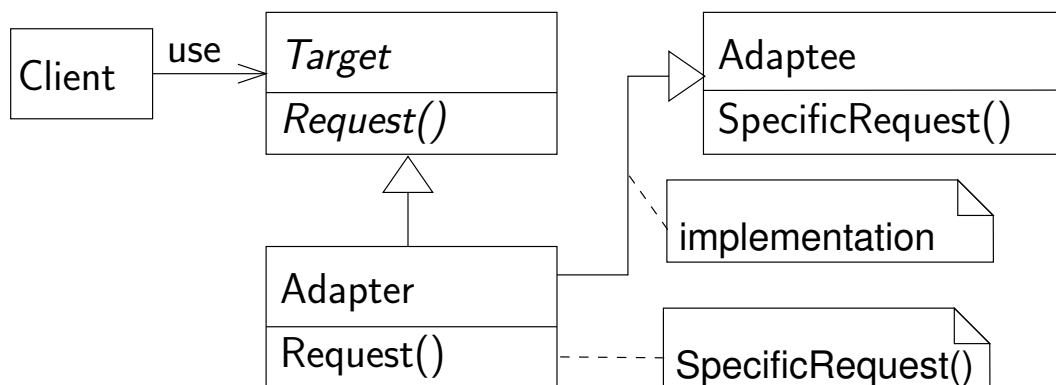


Konsequenzen:

- erlaubt Anpassung von Adaptee und all seiner Unterklassen auf einmal
- Overriding von Adaptees Methoden schwierig
 - Ableitung von Adaptee
 - Adapter referenziert auf Ableitung
- führt Indirektion ein

301 / 504

Lösung: Klassen-Adapter

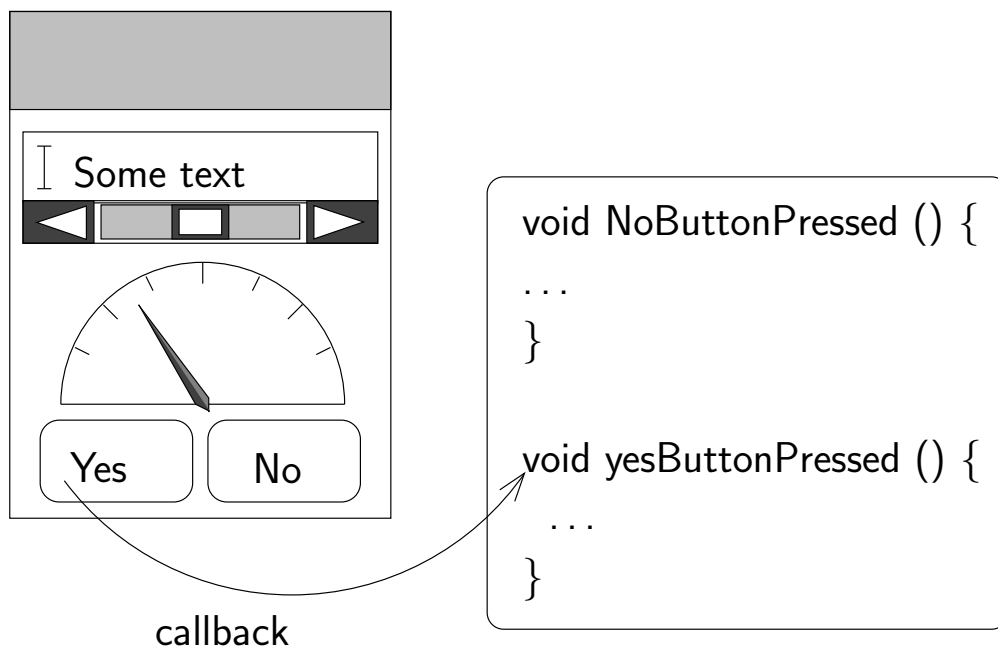


Konsequenzen:

- keine Indirektion
- setzt Mehrfachvererbung voraus
- passt nur Adaptee an, nicht seine Unterklassen
- Overriding von Adaptees Methoden einfach

302 / 504

Problem



303 / 504

Lösung in C: Klient

```
1 void YesButtonPressed () {...}  
2 void NoButtonPressed () {...}  
3  
4 Button mybutton;  
5  
6 int main () {  
7     attach (&mybutton, &YesButtonPressed);  
8     ...  
9     event_loop ();  
10 }
```

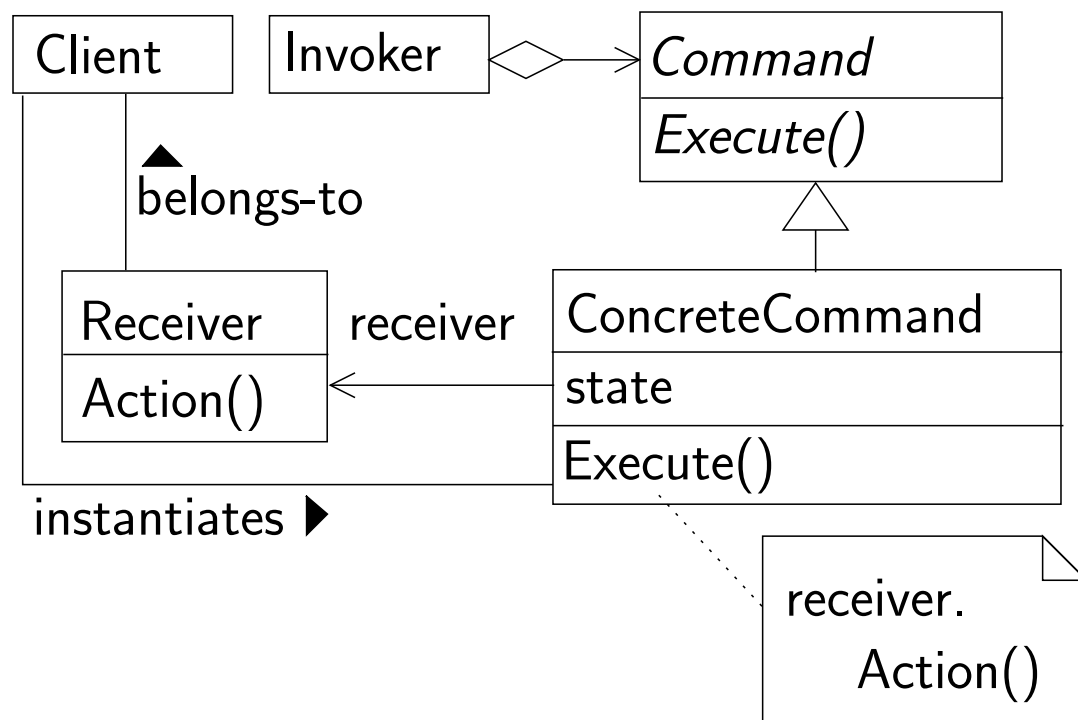
304 / 504

Lösung in C: Mechanismus

```
1 typedef void (*Callback)();
2
3 typedef struct Button {
4     Callback execute;
5     int x;
6     int y;
7 } Button;
8
9 void attach (Button *b, Callback execute) {
10     b->execute = execute;
11 }
12
13 void event_loop () {
14     ...
15     widgets[i]->execute();
16     ...
17 }
```

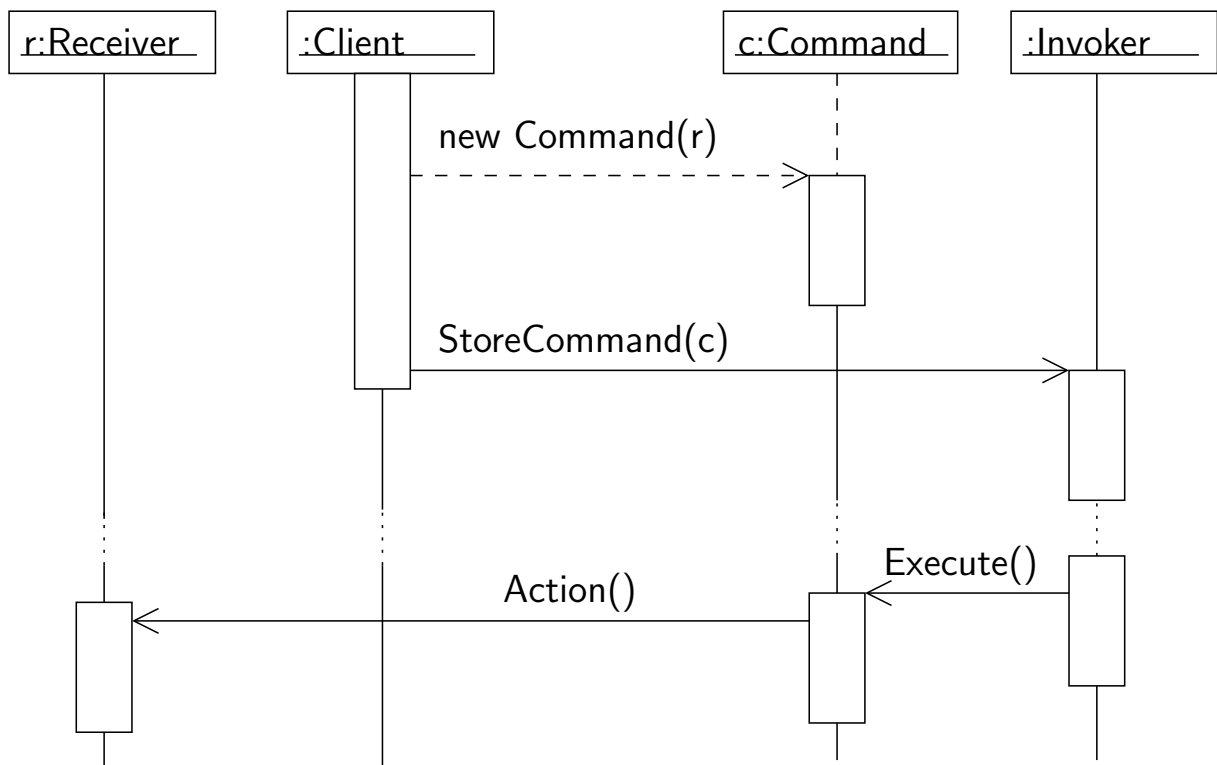
305 / 504

Lösung mit OOP



306 / 504

Lösung mit OOP



307 / 504

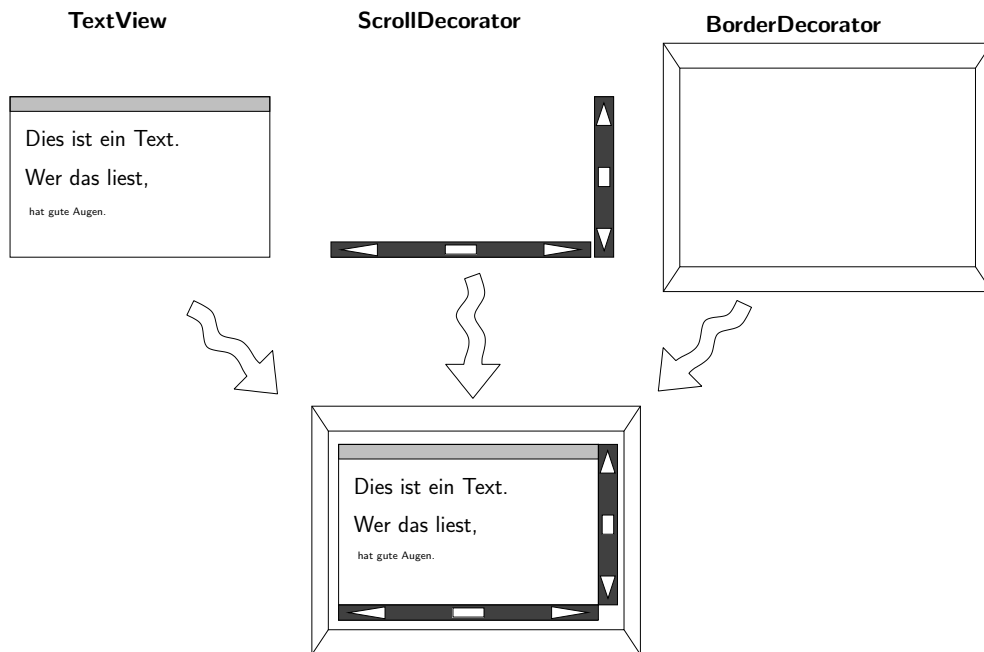
Konsequenzen

- Entkopplung von Objekt, das Operation aufruft, von dem, welches weiß, wie man es ausführt
- Kommandos sind selbst Objekte und können als solche verwendet werden (Attribute, Vererbung etc.)
- Hinzufügen weiterer Kommandos ist einfach

308 / 504

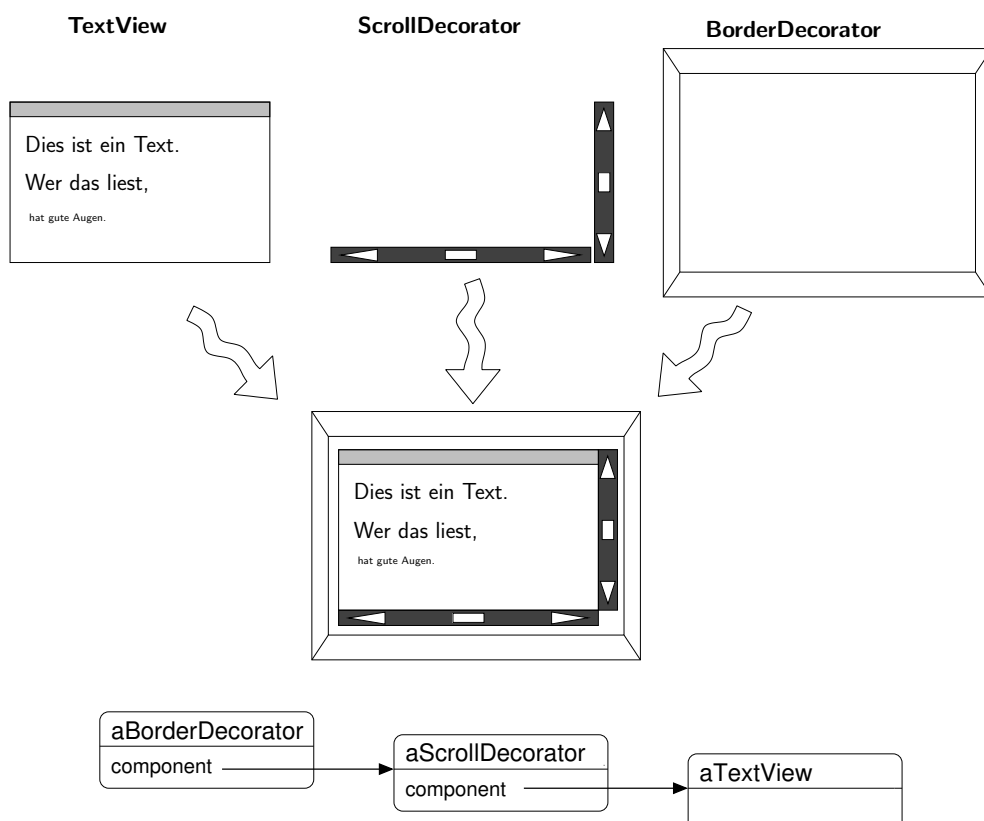
Problem

- Eigenschaften sollen einzelnen Objekten, nicht ganzen Klassen, zugewiesen werden können
- Zuweisung soll dynamisch geschehen



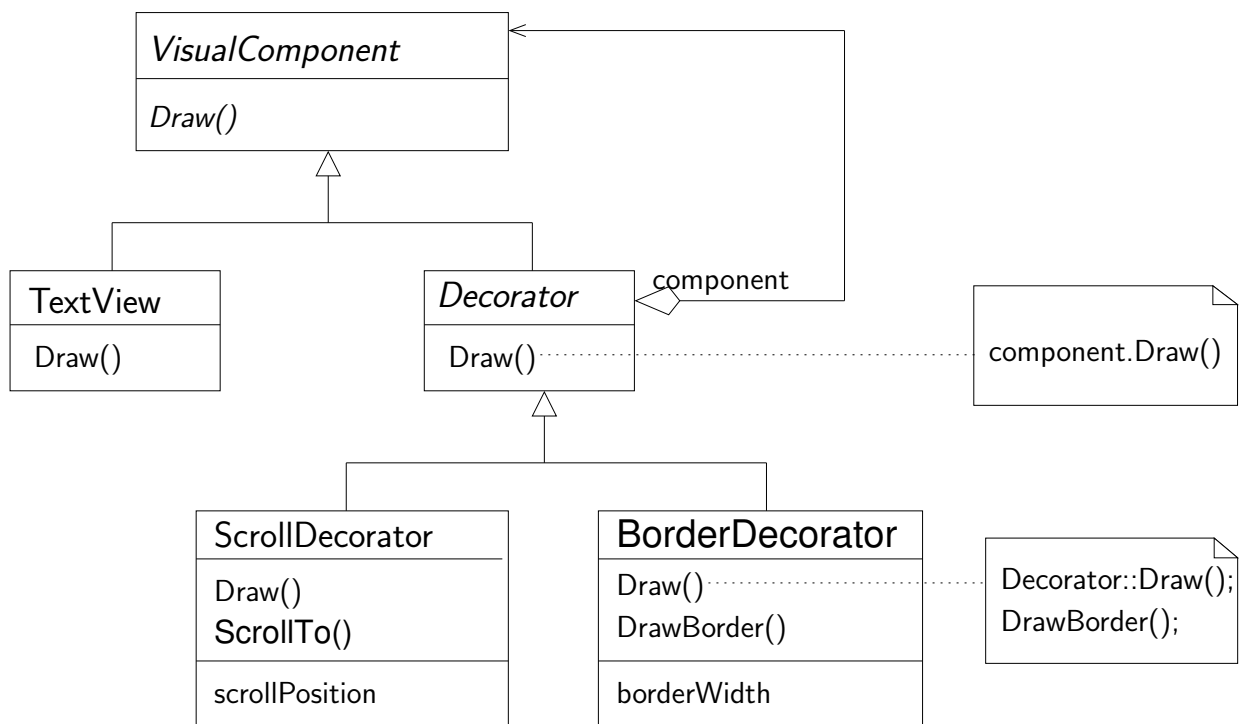
309 / 504

Problem



310 / 504

Lösung



311 / 504

Konsequenzen

- höhere Flexibilität als statische Vererbung
- vermeidet Eier-Legende-Wollmilchsau-Klassen
- Dekorator und dekoriertes Objekt sind nicht identisch
- vielfältige dynamische Kombinationsmöglichkeiten → schwer statisch zu analysieren

312 / 504

Entwurfsmuster Strategy (prozedural)

```
1 typedef enum Strategy {s1, s2, s3} Strategy;  
2 void Apply (Strategy s) {  
3     switch (s) {  
4         case s1: ApplyS1(); break;  
5         case s2: ApplyS2(); break;  
6         case s3: ApplyS3(); break;  
7     }  
8 }
```

313 / 504

Entwurfsmuster Strategy mit OO-Techniken

```
1 class Applier {  
2     Strategy *_strategy  
3  
4     void Apply () {  
5         *_strategy->Algorithm ();  
6     }  
7 }  
8 class Strategy {  
9     virtual void Algorithm () = 0;  
10 }  
11 class Strategy1 : Strategy {  
12     virtual void Algorithm () {...}  
13 }  
14 ...
```

314 / 504

Mutex

```
1 #ifndef MUTEX_H
2 #define MUTEX_H
3 #include "Lock.h"
4 #include <pthread.h>
5 class Mutex : Lock
6 {
7     public:
8         Mutex();           // Creates the Mutex.
9         virtual ~Mutex();  // Destroys the Mutex.
10        void lock();        // Locks the mutex. Blocks if the mutex
11                           // is held by another thread.
12        void unlock();      // Unlocks the mutex so that it can be
13                           // acquired by other threads.
14     private:
15         pthread_mutex_t mutex;
16         friend class Condition;
17 };
18 #endif
```

Locking mit Mutex

```
1 #include "Mutex.h"
2 class Buffer {
3 public:
4     Buffer() : index(-1) {};
5     bool insert(int i) {
6         mutex.lock(); // critical section
7         if(index >= 100) {
8             mutex.unlock();
9             return false;
10        }
11        index++;
12        content[index] = i;
13        mutex.unlock();
14        return true;
15    };
16 private:
17     Mutex mutex;
18     int content[100];
19     int index;
20 };
```

317 / 504

Scoped Locking nach Schmidt u. a. (2000)

```
1 #include "Mutex_Guard.h"
2 class Buffer {
3 public:
4     Buffer() : index(-1) {};
5     bool insert(int i) {
6         // use scoped locking to acquire and release
7         // lock automatically
8         Mutex_Guard guard(mutex);
9         if(index >= 100) {
10            return false;
11        }
12        index++;
13        content[index] = i;
14        return true;
15    };
16 private:
17     Mutex mutex;
18     int content[100];
19     int index;
20 };
```

319 / 504

Das C++ Idiom *Scoped Locking* stellt sicher, dass ein Lock erlangt wird, wenn die Kontrolle einen kritischen Abschnitt betritt, und der Lock wieder frei gegeben wird, wenn der Abschnitt wieder verlassen wird, völlig unabhängig davon, auf welchem Kontrollflusspfad dies geschieht.

– Schmidt u. a. (2000)

Scoped Locking nach Schmidt u. a. (2000)

```
1 #ifndef MUTEX_GUARD_H
2 #define MUTEX_GUARD_H
3 #include "Mutex.h"
4 class Mutex_Guard {
5     public:
6         // store a pointer to the mutex m and acquire it
7         Mutex_Guard(Mutex &m) : mutex(&m), owner(false) {
8             mutex->lock();
9             owner = true; // only true iff lock() succeeds
10        }
11        // release the mutex when guard leaves the scope
12        ~Mutex_Guard() {
13            // only release mutex if it was acquired successfully
14            if(owner) mutex->unlock();
15        }
16        private:
17            Mutex *mutex;
18            bool owner;
19            // disallow copying and assignment
20            Mutex_Guard(const Mutex_Guard&);
21            void operator=(const Mutex_Guard&);
22    };
23 #endif
```

Vorteil: Keine explizite Behandlung von lock und unlock, da impliziter Sprachmechanismus ausgenutzt wird.

Das C++ Idiom *Scoped Locking* stellt sicher, dass ein Lock erlangt wird, wenn die Kontrolle einen kritischen Abschnitt betritt, und der Lock wieder frei gegeben wird, wenn der Abschnitt wieder verlassen wird, völlig unabhängig davon, auf welchem Kontrollflusspfad dies geschieht.

– Schmidt u. a. (2000)

Strategized Locking nach Schmidt u. a. (2000)

```
1 #include "Guard.h"
2 class Buffer {
3 public:
4     Buffer(Lock &l) : lock(&l), index(-1) {};
5     bool insert(int i) {
6         Guard guard(*lock);
7         // critical section
8         // ...
9         return true;
10    };
11 private:
12     Lock *lock;
13     int content[100];
14     int index;
15 };
```

322 / 504

Lock soll verschiedene Implementierungen haben können.
Im Konstruktor von *Buffer* kann die konkrete Implementierung übergeben werden.

Strategized Locking nach Schmidt u. a. (2000)

```
1 #include "Lock.h"
2 class Guard {
3 public:
4     // store a pointer to the mutex m and aquire it
5     Guard(Lock &m) : lock(&m), owner(false) {
6         lock->lock();
7         owner = true; // only true iff lock() succeeds
8     }
9     // release the lock when guard leaves the scope
10    ~Guard() {
11        // only release lock if it was aquired successfully
12        if(owner) lock->unlock();
13    }
14 private:
15     Lock *lock;
16     bool owner;
17     // disallow copying and assignment
18     Guard(const Guard&);
19     void operator=(const Guard&);
20 };
```

323 / 504

Lock soll verschiedene Implementierungen haben können.

Das Entwurfsmuster *Strategized Locking* parameterisiert einen Synchronisationsmechanismus, mit dessen Hilfe die kritischen Abschnitte vor paralleler Ausführung geschützt werden.

– Schmidt u. a. (2000)

Strategized Locking nach Schmidt u. a. (2000)

```
1 #ifndef LOCK_H
2 #define LOCK_H
3 class Lock
4 {
5     public:
6         virtual void lock() = 0;
7         // Acquires the lock. Blocks if the lock
8         // is held by another thread.
9         virtual void unlock() = 0;
10        // Releases the lock so that it can be
11        // acquired by other threads.
12 };
13 #endif
```

324 / 504

Lock ist abstrakte Klasse (Schnittstelle).

Strategized Locking nach Schmidt u. a. (2000)

```
1 #include "Lock.h"
2
3 class NullableLock : Lock
4 {
5     public:
6         NullableLock();
7         ~NullableLock();
8         inline void lock() {}
9         inline void unlock() {}
10 };
```

325 / 504

Mutex implementiert Lock. Falls es keine Threads gibt, dann kann man Nullable verwenden.
Die Entscheidung kann hier zur Laufzeit getroffen werden.
Falls das bereits statisch bekannt ist und man den Laufzeit-Overhead vermeiden möchte, dann kann man Templates benutzen.

Strategized Locking nach Schmidt u. a. (2000)

```
1 #include "Lock.h"
2 template <class LOCK>
3 class Guard_Template {
4 public:
5     // store a pointer to the mutex m and aquire it
6     Guard_Template(LOCK &m) : lock(&m), owner(false) {
7         lock->lock();
8         owner = true;    // only true iff lock() succeeds
9     }
10    // release the lock when guard leaves the scope
11    ~Guard_Template() {
12        // only release lock if it was aquired successfully
13        if(owner) lock->unlock();
14    }
15 private:
16     LOCK *lock;
17     bool owner;
18     // disallow copying and assignment
19     Guard_Template(const Guard_Template&);
20     void operator=(const Guard_Template&);
21 };
```

326 / 504

Hier das Template.

Strategized Locking nach Schmidt u. a. (2000)

```
1 #include "Guard_Template.h"
2 template <class LOCK>
3 class Buffer {
4 public:
5     Buffer() : index(-1) {};
6     bool insert(int i) {
7         Guard_Template<LOCK> guard(lock);
8         // critical section
9         // ...
10        return true;
11    };
12 private:
13     LOCK lock;
14     int content[100];
15     int index;
16 };
```

327 / 504

Der Buffer wird selbst zum Template und kann auf diese Weise statisch parametrisiert werden.

Wie vermeidet man unnötiges Locking und stellt sicher, dass Aufrufe eigener Methoden nicht zu Deadlocks führen?

328 / 504

Deadlock

```
1 #include "Guard.h"
2 class Buffer {
3 public:
4     Buffer(Lock &l) : lock(&l), index(-1) {};
5     bool insert(int i) {
6         Guard guard(*lock);
7         if (size() == 100) return false;
8         else {
9             // ....
10            return true;
11        }
12    };
13     const int size() {
14         Guard guard(*lock);
15         return index + 1;
16     }
17 private:
18     Lock *lock;
19     int content[100];
20     int index;
21 };
```

329 / 504

Thread-Safe Interface nach Schmidt u. a. (2000)

```
1 #include "Guard.h"
2 class Buffer {
3 public:
4     Buffer(Lock &l) : lock(&l), index(-1) {};
5     bool insert(int i) {
6         Guard guard(*lock);
7         return insert_unlocked(i);
8     };
9     const int size() {
10         Guard guard(*lock);
11         return size_unlocked();
12     }
13 private:
14     Lock *lock;
15     int content[100];
16     int index;
17     bool insert_unlocked(int i) {
18         if (size_unlocked() == 100) return false;
19         // ....
20     }
21     const int size_unlocked() { return index + 1; }
22 };
```

330 / 504

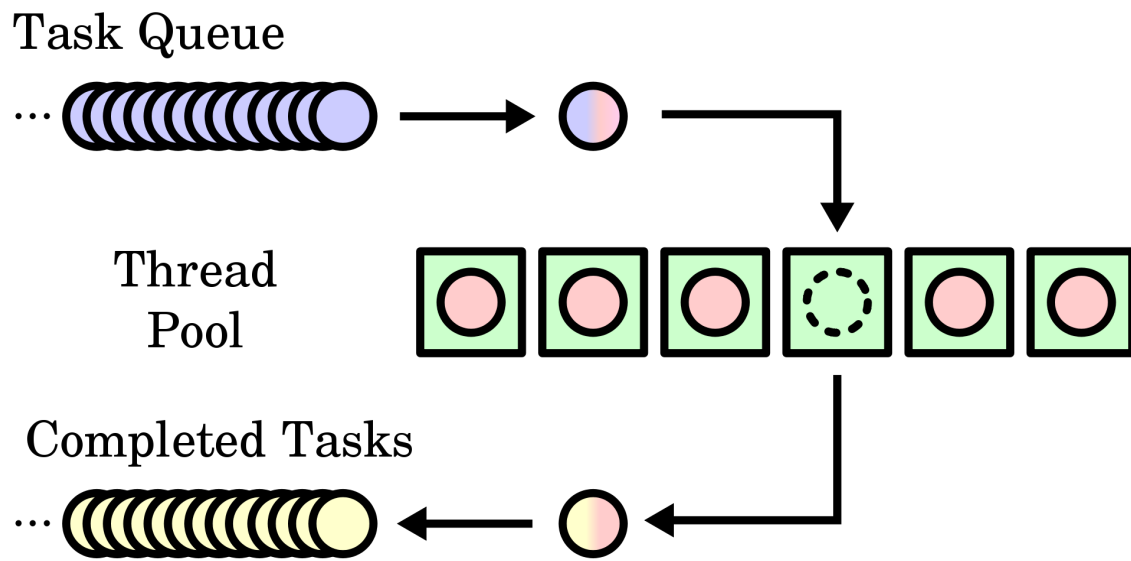
Das Entwurfsmuster *Thread-Safe Interface* vermeidet unnötiges Locking und Deadlocks durch Aufrufe eigener Methoden, indem Synchronisation am Interface von der Implementierung getrennt wird.

Definition

Embarrassingly parallel problem: Ein Problem, das ohne Mühe in parallel abarbeitbare Aufgaben zerlegt werden kann, zwischen denen keine Abhängigkeiten existieren.

Beispiel: Web-Server, der separate Seiten für verschiedene Clients zur Verfügung stellt.

Thread Pool Pattern (auch: Replicated Workers)



332 / 504

A simple thread pool. The task queue has many waiting tasks (blue circles). When a thread opens up in the queue (green box with dotted circle) a task comes off the queue and the open thread executes it (red circles in green boxes). The completed task then "leaves" the thread pool and joins the completed tasks list (yellow circles).

Author: Colin M.L. Burnett

Permission under license: GFDL

Thread Pool Pattern

```
1 class Task {
2     public:
3         void solve();
4         void define(int i);
5     private:
6         int data;
7 };
```

333 / 504

Thread Pool Pattern

```
1 #include "Mutex_Guard.h"
2 #include "Task.h"
3 #include <vector>
4 #include <exception>
5 using namespace std;
6 class Queue {
7     public:
8         Task get() {
9             Mutex_Guard guard(mutex);
10            if(tasks.size()==0) throw exception();
11            Task result = tasks.back();
12            tasks.pop_back();
13            return result;
14        }
15        void put(Task t) {
16            Mutex_Guard guard(mutex);
17            tasks.push_back(t);
18        }
19    private:
20        Mutex mutex;
21        vector<Task> tasks;
22 };
```

334 / 504

Thread Pool Pattern

```
1 Queue queue;
2
3 // entry point of thread
4 void *worker(void *ptr)
5 {
6     while(true) {
7         try
8             { Task t = queue.get();
9               t.solve();
10            }
11         catch(exception &e)
12             { break;}
13     }
14 }
```

335 / 504

Thread Pool Pattern

```
1 main()
2 {
3     // define work
4     for(int i = 0; i < 100; i++) {
5         Task *t = new Task();
6         t->define(i);
7         queue.put(*t);
8     }
9
10    std::vector<pthread_t> thread_pool(10);
11
12    // Create independent threads each of which
13    // will execute function worker.
14    for(int i = 0; i < thread_pool.size(); i++)
15        pthread_create(&thread_pool[i], NULL, worker, NULL);
16
17    // Wait till threads are complete before main continues.
18    for(int i = 0; i < thread_pool.size(); i++)
19        pthread_join(thread_pool[i], NULL);
20 }
```

336 / 504

Was tun, wenn es Produzenten gibt, die kontinuierlich Aufgaben erzeugen?

337 / 504

Beliebig viele Produzenten und Konsumenten

```
1 Queue_Monitor queue(5);
2
3 // entry point of consumer thread
4 void *consumer(void *ptr)
5 {
6     while(true)
7     { Task t = queue.get();
8       t.solve();
9     }
10 }
11
12 // entry point of producer thread
13 void *producer(void *ptr)
14 { Task t;
15   int data = *(int*)ptr;
16   while(true)
17   { t.define(data);
18     queue.put(t);
19     wait(2);
20   }
21 }
```

338 / 504

Monitor Object

```
1 #include "Mutex_Guard.h"
2 #include "Task.h"
3 #include "Condition.h"
4 class Queue_Monitor {
5 public:
6     Queue_Monitor(int capacity);
7     ~Queue_Monitor();
8     Task get();
9     // blocks if no task available
10    void put(Task t);
11    // blocks if no space left
12 private:
13    Mutex mutex;           // monitor lock
14    Condition not_full;    // wait for queue no longer full
15    Condition not_empty;   // wait for queue no longer empty
16    Task *tasks;
17    int size, max_size;
18 };
```

339 / 504

Monitor Condition

```
1 #ifndef CONDITION_H
2 #define CONDITION_H
3 #include <pthread.h>
4 #include "Mutex.h"
5 class Condition
6 {
7 public:
8     Condition(Mutex &m); // Creates the Condition.
9     ~Condition();        // Destroys the Condition.
10    void wait();           // Puts executing thread to sleep
11    void notifyAll();      // Wakes up all sleeping threads.
12 private:
13    pthread_cond_t condition;
14    Mutex &mutex;
15 };
16 #endif
```

340 / 504

Monitor Object I

```
1 #include "Monitor.h"
2
3 Queue_Monitor::Queue_Monitor(int capacity)
4     : size(0), max_size(capacity),
5       not_full(mutex), not_empty(mutex)
6 { tasks = new Task[capacity]; }
7
8 Queue_Monitor::~Queue_Monitor()
9 { delete [] tasks; }
10
11 Task Queue_Monitor::get() {
12     Mutex_Guard guard(mutex);
13     while(size==0) {
14         not_empty.wait();
15     }
16     not_full.notifyAll();
17     return tasks[--size];
18 }
19
20 void Queue_Monitor::put(Task t) {
```

341 / 504

Monitor Object II

```
21     Mutex_Guard guard(mutex);
22     while(size==max_size) {
23         not_full.wait();
24     }
25     not_empty.notifyAll();
26     tasks[size++] = t;
27 }
```

342 / 504

Client-Code

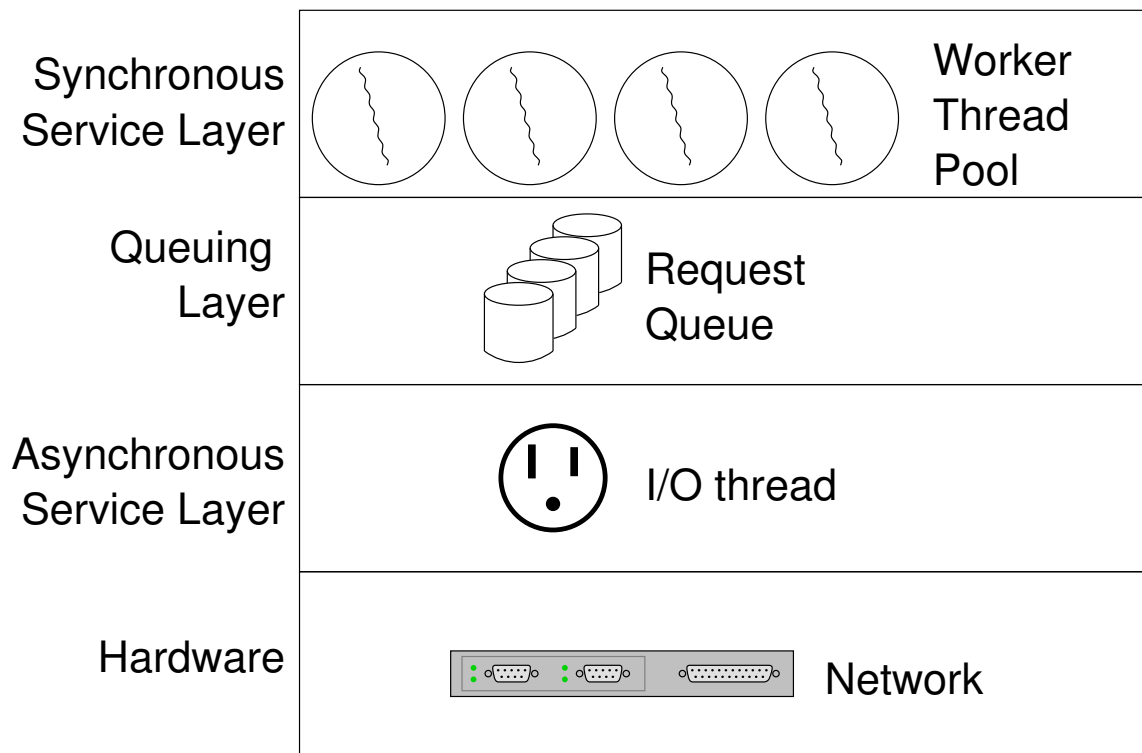
```
1 while(true){
2     // creating a stream (TCP) socket with Poco; it is bound to
3     // a local source port and a network interface address
4     // (hostname, port)
5     Handle socket(Poco::Net::SocketAddress(hostname, port));
6     // send request
7     write(socket, message_type() + toString(i));
8     // get answer
9     std::cout << "reply:_" << read(socket) << std::endl;
10    sleep(1);
11    i++;
12 }
```

344 / 504

Server-Code

```
1 while(true){
2     // waits for any request at all readSockets
3     int ready_sockets
4         = Poco::Net::Socket::select(readSockets, writeSockets,
5                                     exceptSockets, 10000000);
6     // readSockets now contains all sockets where requests are
7     // available; ready_sockets tells how many sockets exist
8     // for which we have requests
9     if (ready_sockets > 0) {
10        // we handle only one of the requests in readSockets;
11        // the others will be served in the next loop iteration
12        Handle connection =
13            ((Poco::Net::ServerSocket)readSockets[0])
14                .acceptConnection();
15        // handle the request by delegating to an EventHandler
16        EventHandler handler(connection);
17        handler.handle_event();
18    }
19 }
```

345 / 504



346 / 504

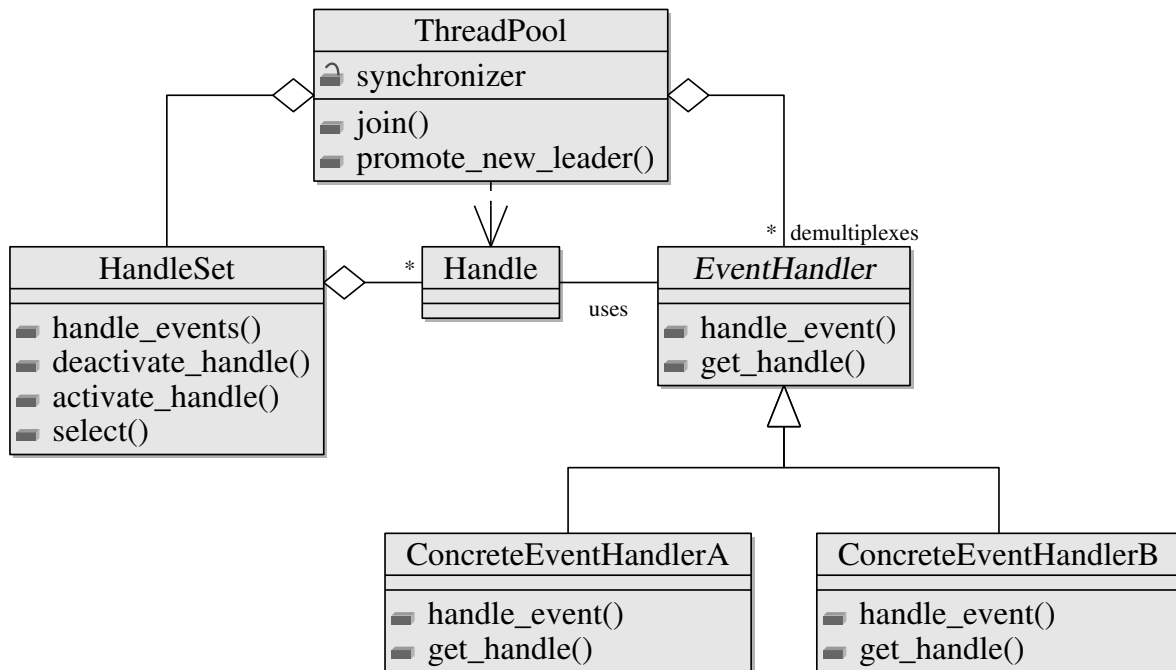
Kontext: Ereignisgesteuerte Anwendung, bei der mehrere Dienstanfragen verschiedener Klienten effizient durch mehrere Threads behandelt werden sollen, die sich die Klienten teilen.

Beispiel: Über ein Netzwerk treffen Anfragen ein, die parallel behandelt werden sollen.

Ein möglicher Entwurf: Ein I/O Thread horcht an einem Socket. Kommt eine Anfrage an, wird sie in eine Queue gesteckt. Die Threads, die die Anfrage bearbeiten, warten, bis Anfragen in der Queue sind, entnehmen diese und bearbeiten sie. Die Queue ist ein Monitor. Der I/O Thread ist ein Produzent und die Dienst-Threads sind Konsumenten.

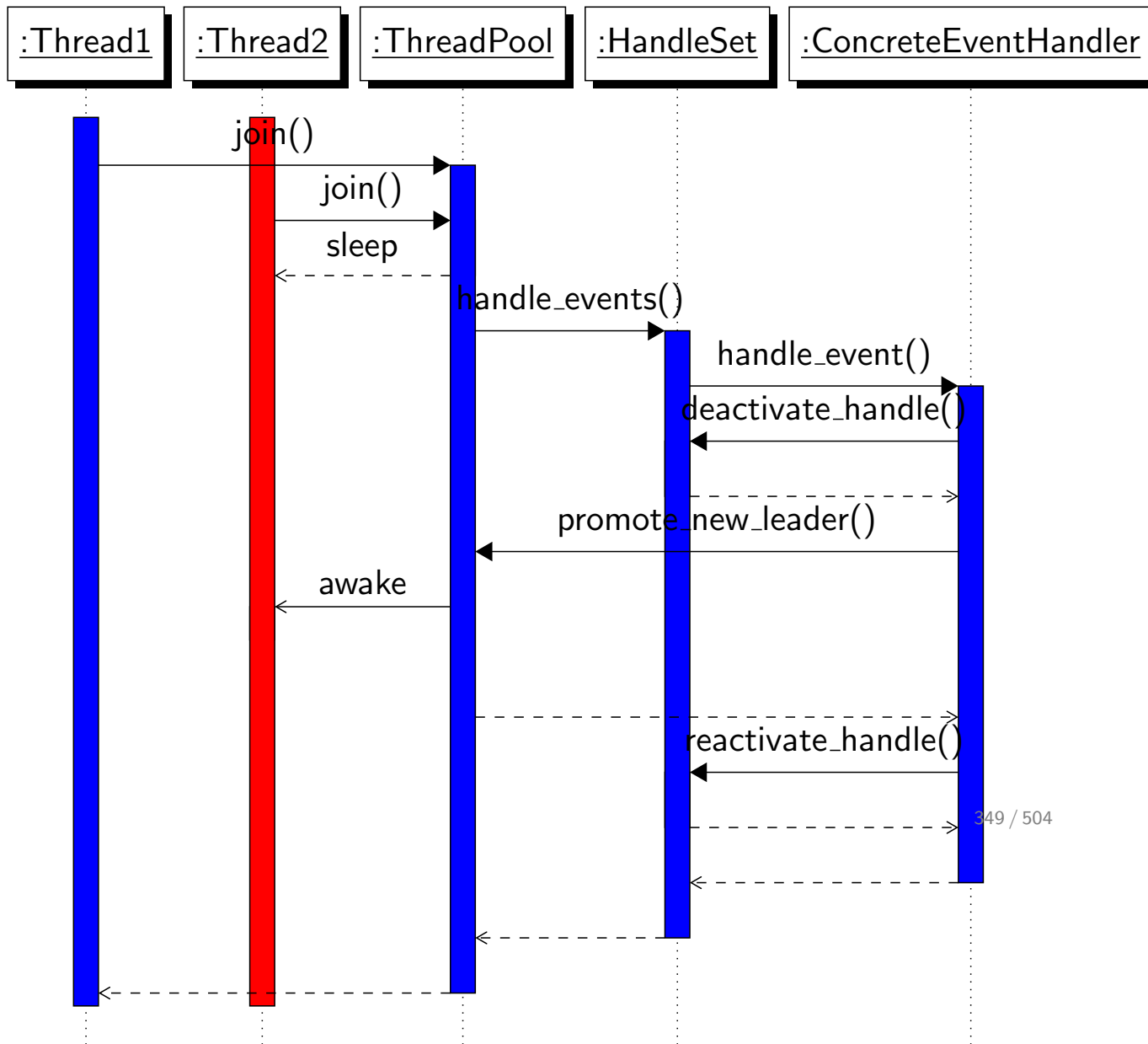
Nachteil: Es findet ein Kontextwechsel zwischen I/O und Dienstbearbeitung statt.

Entwurfsmuster *Leader/Followers* nach Schmidt u. a. (2000)



348 / 504

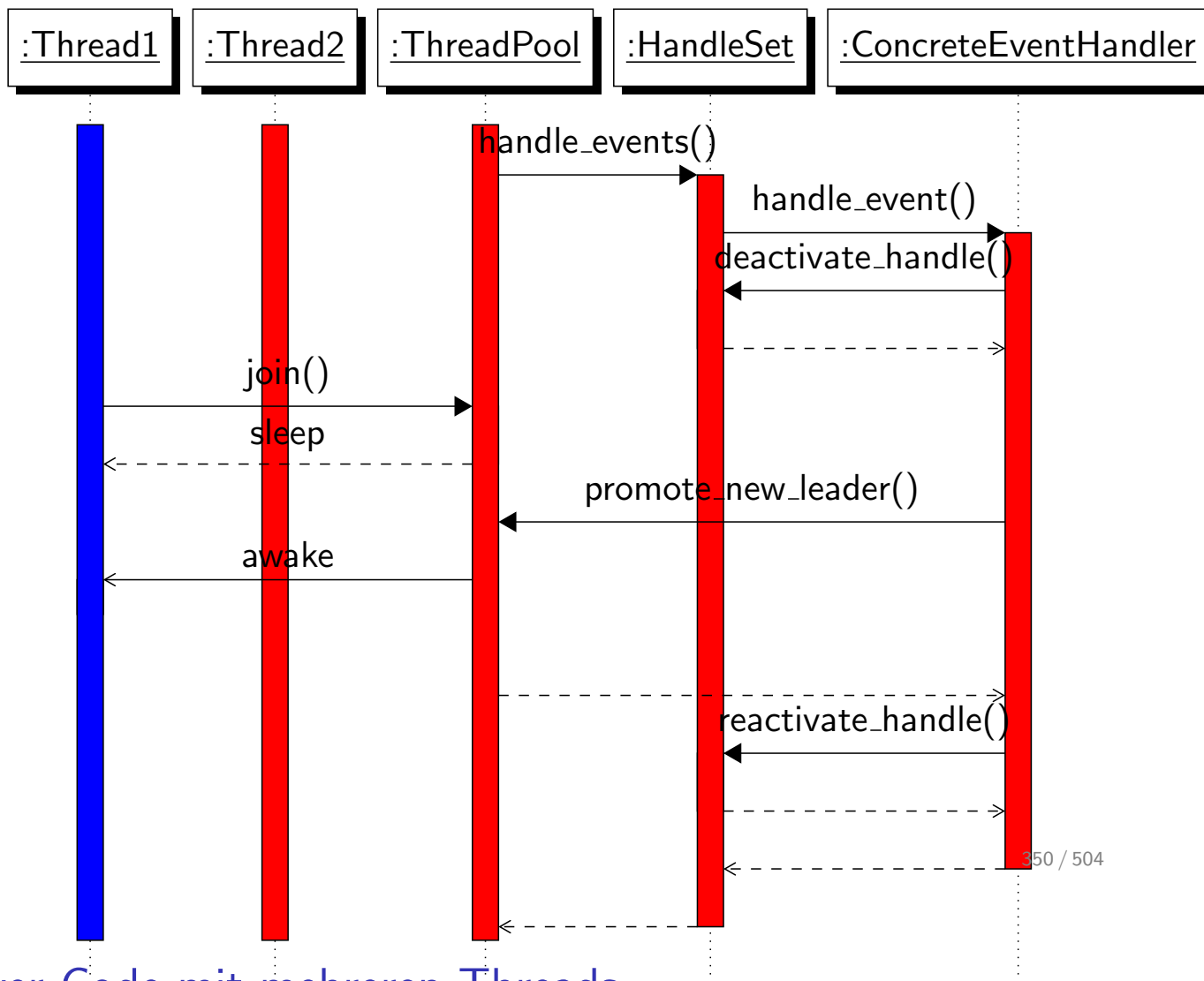
- **Handle**: identifiziert eine Anfrage (ein Ereignis) über Betriebssystemmechanismen wie Netzwerkverbindungen oder geöffnete Dateien.
- **HandleSet**: Menge aller Handles.
- **EventHandler**: Dienst, der für eine Anfrage erbracht werden soll.
- **ThreadPool**: Menge von Threads, die auf Anfragen warten und sie mit Hilfe eines **EventHandler** abarbeiten.
- **Leader**: Thread des **ThreadPool**, der eine Anfrage erhalten hat und sie abarbeitet.
- **Follower**: Threads, die bereit sind, als nächstes zum **Leader** bestimmt zu werden. Bis dahin schlafen sie.



349 / 504

Schritte:

1. Threads melden sich mit `join()` an.
2. Leader wird bestimmt. Andere Prozesse schlafen.
3. Leader wartet auf Ereignis, d.h. auf beliebiges Handle im `HandleSet`.
4. Leader nimmt sich dieses Handle.
5. Leader bestimmt Nachfolger.
6. Ex-Leader führt Auftrag mittels konkretem `EventHandler` aus.
7. Ex-Leader reicht Handle zurück.
8. Ex-Leader tritt mit `join()` dem `ThreadPool` wieder bei.



Server-Code mit mehreren Threads

```

1  // The thread pool where workers will join
2  ThreadPool threadpool(firstport , lastport);
3
4  // Create independent threads
5  std::vector<Poco::Thread*> threads (10);
6  for (int i = 0; i < threads.size(); i++) {
7      threads[i] = new Poco::Thread();
8  }
9
10 std::vector<Worker*> workers;
11 // each of which will execute function Worker::run().
12 for (int i = 0; i < threads.size(); i++) {
13     Worker* worker = new Worker(&threadpool);
14     workers.push_back(worker);
15     threads[i]->start(*worker);
16 }
17
18 // wait until all threads are finished
19 for (int i = 0; i < threads.size(); i++) {
20     threads[i]->join();
21 }

```

Worker-Thread

```
1 class Worker : public Poco::Runnable {
2 public:
3     Worker(ThreadPool* pool) : threadpool(pool) {};
4     virtual ~Worker() {};
5     void run() {
6         // endless loop, stopped only by killing the process
7         while(true) {
8             threadpool->join(id);
9         }
10    }
11 private:
12     ThreadPool* threadpool;
13 };
```

352 / 504

Thread-Pool

```
1 class ThreadPool {
2 public:
3     ThreadPool(Poco::UInt16 firstport , Poco::UInt16 lastport);
4
5     void join();
6     // to join the thread pool to become a follower or leader;
7
8     void promote_new_leader();
9     // a new leader is selected; this method must be called
10    // only by the current leader
11 private:
12     Poco::FastMutex mutex;
13     // used to block joining threads that are followers
14
15     HandleSet handleset;
16 };
```

353 / 504

Thread-Pool

```
1 ThreadPool::ThreadPool(Poco::UInt16 firstport ,
2                       Poco::UInt16 lastport)
3     : handleset(this , firstport , lastport)
4     {};
5
6 void ThreadPool::join()
7 {
8     mutex.lock();
9     // if we arrive here, the executing thread
10    // becomes the leader
11    handleset.handle_events();
12 }
13 void ThreadPool::promote_new_leader()
14 {
15     mutex.unlock();
16 }
```

354 / 504

Handle-Set

```
1 class HandleSet {
2 public:
3     HandleSet(ThreadPool* pool ,
4               Poco::UInt16 firstport ,
5               Poco::UInt16 lastport);
6     void handle_events();
7 private:
8     Poco::Net::Socket::SocketList handles;
9     ThreadPool* threadpool;
10    // the list of handles (sockets) to be listened
11    void deactivate_handle(Handle handle);
12    void activate_handle(Handle handle);
13    Handle select();
14 };
```

355 / 504

```

1 HandleSet::HandleSet(ThreadPool* pool,
2                       Poco::UInt16 firstport,
3                       Poco::UInt16 lastport)
4     : threadpool(pool)
5 {
6     // bind to all sockets
7     for (int i = firstport; i <= lastport; i++) {
8         //creating a socket with Poco
9         // a ServerSocket is intended for servers
10        Poco::Net::ServerSocket socket(i);
11        handles.push_back(socket);
12    }
13 };
14
15 void HandleSet::deactivate_handle(Handle handle) {}
16 // nothing to be done
17
18 void HandleSet::activate_handle(Handle handle) {}
19 // nothing to be done

```

356 / 504

```

1 void HandleSet::handle_events() {
2     // obtain the handle
3     Handle handle = select();
4
5     // this is the handler that handles the event
6     EventHandler handler(handle);
7
8     deactivate_handle(handle);
9
10    // we deviate from the leader/follower pattern here
11    // and promote the new leader right here and not
12    // in the EventHandler
13    threadpool->promote_new_leader();
14
15    // handle the request by delegating to the EventHandler
16    handler.handle_event();
17
18    activate_handle(handle);
19 }

```

357 / 504

```

1 // awaits and handles a request listening on all sockets
2 // in readSockets
3 Handle HandleSet::select() {
4     // local copy of handles needed because select overwrites
5     // its arguments
6     Poco::Net::Socket::SocketList readSockets = handles;
7
8     // waits for any request at all readSockets;
9     // endless loop because of timeout
10    while(true){
11        int ready_sockets
12            = Poco::Net::Socket::select
13              (readSockets, writeSockets, exceptSockets, 100000);
14        // readSockets now contains all sockets where requests are
15        // available ready_sockets tells how many sockets exist for
16        // which we have requests
17        if (ready_sockets > 0) break;
18    }
19    // we handle only one of the requests in readSockets;
20    // the others will be served later by the next thread;
21    // return the handle (connection) for that request
22    return ((Poco::Net::ServerSocket)readSockets[0])
23           .acceptConnection();
24 }

```

358 / 504

Weiterführende Literatur

- Das Standardbuch zu Entwurfsmustern ist das von Gamma u. a. (2003)
- Die Muster für Synchronisation und Parallelisierung stammen von Schmidt u. a. (2000); in dieser Reihe sind weitere Bücher zu Mustern erschienen.

Wiederholungsfragen

- Was ist ein Entwurfsmuster?
- Warum sind sie interessant für die Software-Entwicklung?
- Wie werden Entwurfsmuster von Gamma et al. kategorisiert?
- Erläutern Sie ein in der Vorlesung vorgestellten Entwurfsmuster X (mit Vor- und Nachteilen und Variationen; dabei wird X vom Prüfer vorgegeben).

360 / 504

Komponentenbasierte Entwicklung

Komponentenbasierte Entwicklung

Lernziele

Komponente

Komponentenbasierte Entwicklung

Komponentenmodelle

Schnittstellen

Beschaffung und Entstehung

Herstellung

Implementierungsaspekte

Architektur-Mismatch

Wiederholungsfragen

361 / 504

Lernziele

- Benutzung von Komponenten
 - Komponentenbasierte Entwicklung
 - Komponentenmodelle
 - Schnittstellen
 - Komponenten
 - Zusammenbau
- Erschaffung von Komponenten
 - Herstellung
 - Implementierungsaspekte

363 / 504

Komponente

Definition

Software components: are executable units of independent production, acquisition, and deployment that interact to form a functioning system (Szyperski u. a. 2002).

- “to compose a system”
- dazu da, um zusammengesetzt zu werden
- N.B.: Komponente $\neq$ Klasse $\neq$ Verteilung

364 / 504

Komponentenbasierte Entwicklung

- Trennung von Schnittstellen (liefern, benutzen) und Implementierung
 - Standards für Integration
 - einheitliche Schnittstellenbeschreibung
 - unabhängig von der Programmiersprache
 - Infrastruktur (Middleware)
 - Entwicklungsprozess
- technische und nicht-technische Aspekte

365 / 504

Motivation I

Wiederverwendung als Schlüssel:

- ökonomisch
 - als Lösung der Softwarekrise
 - OO konnte die Erwartungen an Wiederverwendbarkeit und Vermarktung nicht erfüllen
- ohne Wiederverwendung: nur lineares Wachstum möglich
- mit Wiederverwendung: superlineares Wachstum möglich (so die Hoffnung)

366 / 504

Ebenen der Wiederverwendung

- konkrete Lösungsteile: Bibliotheken
- Verträge: Schnittstellen
- Vertragsanbieter: Komponenten
- einzelne Interaktionsteile: Meldungen und Protokolle
- Architekturen für Interaktion: Muster
- Architekturen für Teilsysteme: Frameworks
- Gesamtsystem: Systemarchitekturen

367 / 504

Maßgefertigt vs. Standardsoftware

- **Maßanfertigung:**
 - optimale Anpassung an Kunde → Wettbewerbsvorteil
 - keine Änderung der Kundenprozesse notwendig
 - unter lokaler Kontrolle
- **Standardsoftware:**
 - billiger
 - schneller einsetzbar
 - geringeres Risiko des Scheiterns
 - Wartung und Evolutionsanpassung durch den Hersteller
 - leichtere Zusammenarbeit mit anderen Systemen

Vorteile von beiden Ansätzen: Maßanfertigung aus Standardkomponenten

368 / 504

Integration

Softwerker werden bei komponentenbasierter Entwicklung zu Systemintegratoren:

- Integrationsproblematik wird verschärft:
 - Einbau der Komponenten von Dritten
 - Komponenten kommen aus verschiedenen Quellen
 - Fehlereingrenzung schwierig
 - keine klassischen Integrationstests, da späte Integration
 - System nur so stark wie schwächste Komponente
- defensives Programmieren (bei Hersteller und Verwender) ist zu empfehlen

369 / 504

Vor- und Nachteile I

Vorteile:

- verbesserte Qualität
- Wiederverwendung verringert die Dauer bis zur Auslieferung
- Modularisierung (nur lokale Änderungen, Abhängigkeiten explizit, ...)
- Austauschbarkeit durch Abstraktion (Schnittstellen)
- Markt: innovative Produkte, niedriger Preis,...

370 / 504

Vor- und Nachteile II

Nachteile:

- höhere Kosten durch:
 - komplexere Technik
 - durch Outsourcing höhere Kosten durch Risikoabstützung
 - mehr Aufwand, um vergleichbare Systemstabilität zu erreichen
- offene Probleme:
 - Vertrauenswürdigkeit der Implementierung
 - Zertifizierung der Implementierung
 - gewünschte Anforderungen vs. verfügbare Komponenten

371 / 504

Prozess

Anpassung des Entwicklungsprozesses:

- 1 Anforderungen erfassen
- 2 Identifizierung von Komponentenkandidaten (suchen, auswählen, überprüfen)
- 3 Anpassen der Anforderungen an gefundene Kandidaten
- 4 Design der Architektur
- 5 Überprüfung der Komponentenkandidaten und event. neue Suche
- 6 Erstellen der nichtabgedeckten Funktionalität
- 7 Zusammenstellen des Systems aus Komponenten mit Verbindungscode (Glue-Code)

372 / 504

Komponentenmodelle

- Sicherstellen der Interoperabilität
- Standards für
 - Schnittstellen:
 - Schnittstellenbeschreibung
 - spezielle Schnittstellen
 - Interaktion zwischen Komponenten
 - Informationen zur Verwendung:
 - Namensregeln
 - Individualisieren
 - Zugriff auf Metadaten
 - Einsatz:
 - Verpackung
 - Dokumentation
 - Evolutionsunterstützung

376 / 504

Beispiele für Komponentenmodelle

- im weiteren Sinn:
 - Anwendungen in einem Betriebssystem
 - Plugins
 - Verbunddokumente (Office Dokumente mit OLE, HTML)
- im engeren Sinn: CORBA, COM, JavaBeans, OSGI

377 / 504

Eigenschaften erfolgreicher Komponentenmodelle

- Infrastruktur mit guter Basisfunktionalität
- Komponenten werden von unterschiedlichen Herstellern angeboten und von Kunden eingesetzt
- Komponenten von verschiedenen Anbietern arbeiten in einer Installation zusammen
- Komponenten haben eine Bedeutung für den Klienten

379 / 504

Allgemeine Dienste

- meist durch Komponentenmodelle als Schnittstelle festgelegt
- Anbieter der Komponentenmodelle bietet auch diese Komponenten an
- besonders wichtig für verteilte Systeme
- Beispiele:
 - Verzeichnisdienst
 - Persistenz
 - Nachrichtendienst
 - Transaktionsmanagement
 - Sicherheit

380 / 504

Schnittstellen

- Schnittstellen ermöglichen:
 - Zusammenarbeit zwischen fremden Komponenten
 - Austauschbarkeit (Anbieter und Benutzer)
 - Identifizierung der Abhängigkeiten
- Interna bleiben verborgen (alle Zugriffe über Schnittstelle)
- Qualität von höchster Bedeutung
- eine Komponente kann Serviceprovider für mehr als eine Schnittstelle sein (provides)
- eine Komponente kann den Service anderer Schnittstellen benötigen (requires)
- späte Integration → späte Bindung → Indirektion
- beschrieben durch Meta-Information zur Laufzeit, Interface Description Language (IDL) oder „direkt“ in Programmiersprache

382 / 504

Beispiel (CORBA-IDL)

```
module Bank {
    typedef long pin_t;
    enum KontoFehlerTyp {
        UngenuegendeKontodeckung
    };

    exception KontoException {
        KontoFehlerTyp typ;
        string beschreibung;
    };

    interface Konto {
        readonly attribute string name;
        readonly attribute long kontoStand;

        boolean isValidPin(in pin_t pin);
        void abheben(in long betrag) raises(KontoException);
        void einzahlen(in long betrag);
    };
};
```

383 / 504

Vertrag

- Schnittstelle als Vertrag zwischen Anbieter und Benutzer des Services
- Anbieter:
 - über Funktionalität: z.B. als Vor- und Nachbedingungen
 - über nicht-funktionale Anforderungen (Service-Level, Ressourcen);
z.B. Standard Template Library für C++
 - Darstellung:
 - informal als Text
 - formaler z.B. durch temporale Logik (um Terminierung zuzusichern) oder mit OCL
- Kunde:
 - Vermeidung von speziellen Eigenschaften einer bestimmten Implementierung (d.h. nur Vertrag benutzen)

384 / 504

Versionen

- Problem: sowohl Schnittstelle als auch Implementierung ändern sich → unterschiedliche Versionen nicht vermeidbar
- Ziel: Entscheidung, ob kompatibel oder nicht
 - zusätzlich noch Unterstützung für einen Bereich von Versionen (sliding window)
- Lösungen (Lösungen?):
 - unveränderliche Schnittstellen
 - Schnittstellen dürfen sich ändern, aber nur nach Regeln (z.B. Parametertyp darf verallgemeinert werden)
 - Ignorieren des Problems:
 - abwälzen auf tiefere Schicht
 - immer neu kompilieren

385 / 504

- Komponenten werden nach funktionalen und nichtfunktionalen Anforderungen klassifiziert
- Qualitätskriterien für Auswahl:
 - Erfüllungsgrad funktionaler und nichtfunktionaler Anforderungen
 - Schnittstellenbeschreibung
 - Abhängigkeiten und Kompatibilität
 - Vertrauenswürdigkeit, Überlebenschance des Anbieters, Garantien und Wartungszusicherung
 - Preis und Zahlungsart
 - ...

387 / 504

Entstehung von Komponenten

- meist aus bestehender Software
- Anpassung notwendig, um Wiederverwendbarkeit zu erreichen
- ist Investition ökonomisch sinnvoll?
 - Größe des Einsatzgebiets
 - bislang angebotene Funktionalität
 - potentieller Grad der Wiederverwendung

Balance zwischen

großen Komponenten

- bieten mehr Service an
- weniger Abhängigkeiten
- schneller, da keine cross-context Aufrufe

kleinen Komponenten

- verständlicher
- billiger für den Benutzer
- mehr Freiheit für den Benutzer
- mehr Benutzer

388 / 504

Implementierung

- defensives Programmieren, da keine Integrationstests
- existierenden Code wiederverwendbar machen:
 - entferne anwendungsspezifische Methoden
 - generalisiere Namen
 - füge Methoden hinzu, um Funktionalität zu vervollständigen
 - führe konsistente Ausnahmebehandlung ein
 - füge Möglichkeiten hinzu, die Komponenten an verschiedene Benutzer anzupassen
 - binde benötigte Komponenten ein, um Unabhängigkeit zu erhöhen

390 / 504

Typen

- Typ als vereinfachter Vertrag
- syntaktische Überprüfung durch Compiler
- semantische Überprüfung durch explizite Vor- und Nachbedingungen (Übersetzungszeittest besser als Ladezeittest besser als Laufzeittest besser als kein Test)
- Implementierungen dürfen Vorbedingungen abschwächen oder Nachbedingungen verschärfen

391 / 504

Vererbung

- Anpassen einer Komponente:
 - durch Parametrisieren und Verbinden mit anderen Komponenten
 - durch Ableiten von Subtypen
- Arten:
 - Implementierungsvererbung
 - Schnittstellenvererbung
- Subtypen
 - Austauschbarkeit (Liskovsches Substitutionsprinzip); deklarative vs. strukturelle Subtypen
 - Vererbung von Parametertypen in Signatur `foo(T)` in Klasse `T`:
 - Invarianz: selber Typ `T`
 - Kontravarianz: Supertyp von `T`
 - Kovarianz: Subtyp von `T`
- Mehrfachvererbung:
 - Schnittstellenvererbung: kein Problem
 - Implementierungsvererbung: problematisch

392 / 504

Zerbrechliche Basisklassen

- Basisklasse wird geändert, sind erbende Klassen immer noch funktionsfähig?
- unvorhergesehene Aufrufgraphen

// Version 1

```
class T {  
    public void foo() {}  
}
```

// Client code

```
class NT {  
    public void bar() {}  
}
```

// Version 2

```
class T {  
    public void foo() {bar();}  
    private int i;  
    public void bar() { i = 1;}  
}
```

393 / 504

Zerbrechliche Basisklassen

- notwendig ist Schnittstellenvertrag zwischen Klasse und ererbenden Klassen
- Lösungen:
 - Ableiten der Klasse verbieten (z.B. `final`)
 - Sichtbarkeitsschutz (`public`, `protected`, `private`, `final`, `override`)
 - Offenlegung der Funktionsweise der Basisklassen und Regeln, was Unterklassen dürfen

394 / 504

Aufrufe zwischen Komponenten in verteilten Systemen

- Idee des lokalen Aufrufes bleibt erhalten
- Generierung von Stubs
- Aktionen des Aufrufers bei Aufruf:
 - 1 Kontrolle geht an Stub
 - 2 Marshalling/Serialisierung
 - 3 Übertragung der Parameter
 - 4 Ausführen auf dem Ziel
 - 5 Übertragung des Ergebnisses/Ausnahme
 - 6 Unmarshalling
 - 7 Rückgabe an den Aufrufer
- Optimierung für lokalen Fall

395 / 504

Wiedereintritt

```
int global = 1;

int foo()
{// non reentrant
  global = global + 1; // what if interrupted here?
  return global;
}

int bar()
{// non reentrant (transitivity)
  return foo() + 1;
}
```

- Vorkommen: Callbacks (von der unteren zur oberen Schicht) oder Multithreading
- Problem: welche Funktionen dürfen wann benutzt werden?
Beispiel: Dokumentation zu Unix-Signal-Handler
- nicht mit Vor- und Nachbedingungen ausdrückbar

396 / 504

Fallstudie zur komponentenbasierten Entwicklung

Garlan u. a. (1995): Komposition eines Case-Tools aus

- einer objektorientierten Datenbank
- Toolkit zur Konstruktion graphischer Benutzeroberflächen
- event-basiertem Tool-Integrations-Mechanismus
- RPC-Mechanismus

Alle Komponenten in C oder C++ geschrieben.

397 / 504

Glaube und Wirklichkeit

Schätzung:

- Dauer: 6 Monate
- Aufwand: 12 PM

Tatsächlich:

- Dauer: 2 Jahre
- Aufwand: 60 PM für ersten Prototyp

Ergebnis:

- sehr großes System
- träge Performanz
- viele Anpassungen für die Integration notwendig
- existierende Funktionalität musste neu implementiert werden, weil sie nicht exakt den Anforderungen entsprach

398 / 504

Architektur-Mismatch

Definition

Architektur-Mismatch: inkompatible Annahmen von wiederzuverwendenden Komponenten über das Systems, in dem sie eingesetzt werden sollen.

Meist spät entdeckt, weil die Annahmen in aller Regel nur implizit sind.

399 / 504

Komponenten betreffend

Annahmen von Komponenten über

- ihre Infrastruktur, die zur Verfügung gestellt werden soll bzw. vorausgesetzt wird
 - Komponenten stellten vieles zur Verfügung, was gar nicht gebraucht wurde
 - exzessiver Code
- das Kontrollmodell: wer steuert den Kontrollfluss
 - jede Komponente nahm an, dass sie die Hauptkontrollschleife darstelle
 - aufwändige Restrukturierung notwendig
- das Datenmodell: Art und Weise, wie die Umgebung Daten manipuliert, die von der Komponente verwaltet werden
 - hierarchische Datenstruktur erlaubte Änderung der Teile nur über das Ganze
 - für Anwendung zu unflexibel
 - teilweise Neuimplementierung

400 / 504

Konnektoren betreffend

Annahmen von Konnektoren über

- Protokoll: Interaktionsmuster
 - Semantik des synchronen Aufrufs passte nicht
 - Ausweichung auf RPC des Betriebssystems hierfür
- Datenmodell: Art der Daten, die kommuniziert werden
 - RPC des Betriebssystems nahm an, C-Datenstrukturen zu transportieren
 - wiederzuverwendendes Event-Broadcast-System nahm an, ASCII zu transportieren
 - Konvertierungsroutinen wurden notwendig

401 / 504

Architekturkonfiguration betreffend

Annahmen über Architekturkonfiguration

- Topologie der Systemkommunikation
 - Datenbank nahm an, dass verbundene Tools nicht kooperieren wollen und blockierte sie, um Sequenzialisierung zu garantieren
 - Tools mussten aber kooperieren
 - eigener Transaktionsmonitor musste implementiert werden
- An- oder Abwesenheit bestimmter Komponenten und Konnektoren

402 / 504

Konstruktionsprozess betreffend

Annahmen über Konstruktionsprozess (wie Komponenten/Konnektoren aus generischen Einheiten erstellt werden – sowohl zur Übersetzungs- als auch zur Laufzeit)

Beispiele:

- Datenbank → Schema muss festgelegt werden
- Event-System → Menge der Ereignisse und Registration

Annahmen über die Reihenfolge, in der Teile erstellt und kombiniert werden.

- nicht zueinander passende Annahmen über den Konstruktionsprozess
 - Umwege machten Konstruktionsprozess aufwändig und kompliziert

403 / 504

Wiederholungs- und Vertiefungsfragen

- Was ist eine Komponente?
- Was ist die Idee der komponentenbasierten Entwicklung?
Welche Ziele verfolgt sie?
- Diskutieren Sie Vor- und Nachteile maßgefertigter Software versus Software von der Stange.
- Was ist ein Komponentenmodell? Was wird von einem solchen üblicherweise festgelegt bzw. zur Verfügung gestellt?
- Was ist eine Schnittstelle und welche Bedeutung kommt diesem Konzept im Kontext komponentenbasierter Entwicklung zu?
- Wie können vorgefertigte Komponenten angepasst werden?
- Welches Problem tritt bei Anpassung durch Vererbung und Redefinition auf? Wie kann man mit ihm umgehen?
- Was ist das Problem des Wiedereintritts?
- Was versteht man unter Architektur-Mismatch und welche Bedeutung hat er für die komponentenbasierte Entwicklung?

404 / 504

OSGI

OSGI als Beispiel eines Komponentenmodells

Architektur

Manifest

Lebenszyklus von Bundles

Existierende Container-Implementierungen

Demo mit Eclipse

 Einfaches Bundle anlegen

 OSGI-Konsole und Lebenszyklus

 Kooperierende Bundles

 Service-Orientierung mit Bundles

Wiederholungsfragen

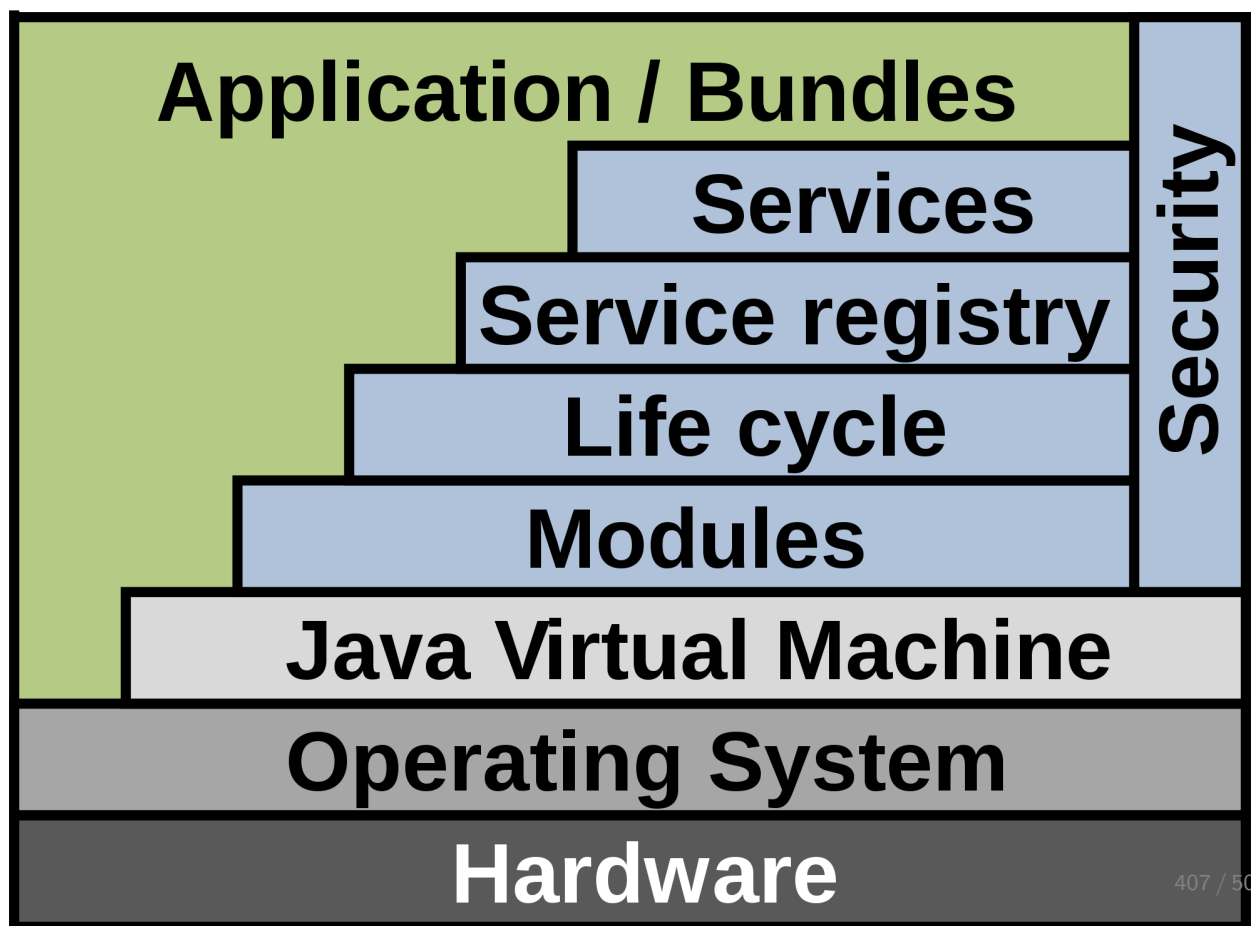
Beispiel eines Komponenten-Rahmewerks: OSGI

Open Services Gateway Initiative (OSGI)

- Komponentenmodell für Java
- Komponente = *Bundle* = JAR + Manifest
- unterstützt entfernte Installation, Start, Stopp, Aktualisierung und Deinstallation von *Bundles*
- Service-Register ermöglicht den Bundles, Dienste hinzuzufügen, zu entfernen und anzupassen
- Verbreitung: Eclipse, Mobiltelefone, ...

406 / 504

Architektur

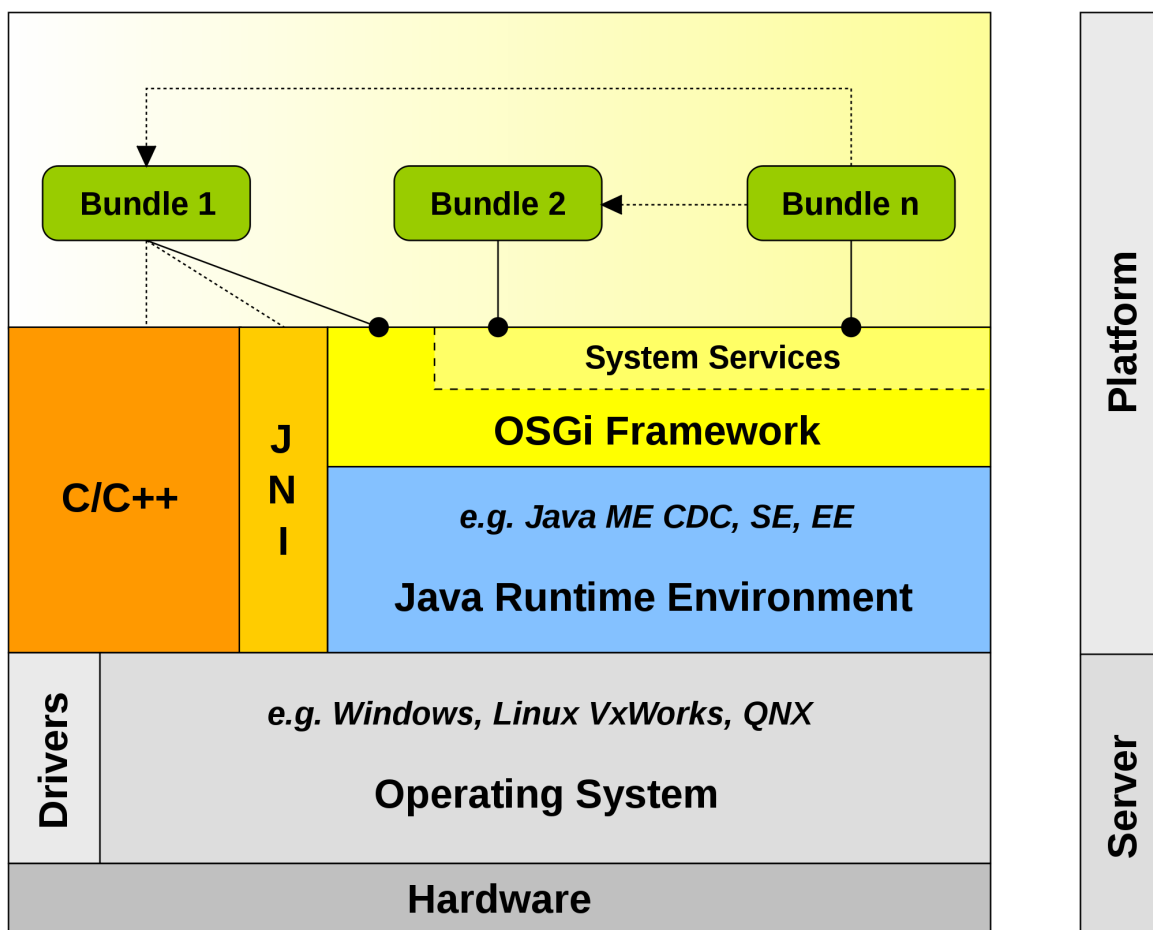


407 / 504

Bundles	Bundles are normal jar components with extra manifest headers.
Services	The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java Interfaces (POJI) or Plain Old Java Objects POJO.
Services Registry	The API for management services (ServiceRegistration, ServiceTracker and ServiceReference).
Life-Cycle	The API for life cycle management for (install, start, stop, update, and uninstall) bundles.
Modules	The layer that defines encapsulation and declaration of dependencies (how a bundle can import and export code).
Security	The layer that handles the security aspects by limiting bundle functionality to pre-defined capabilities.

Bildquelle: Faisal Akeel, Bill Streckfus

Architektur

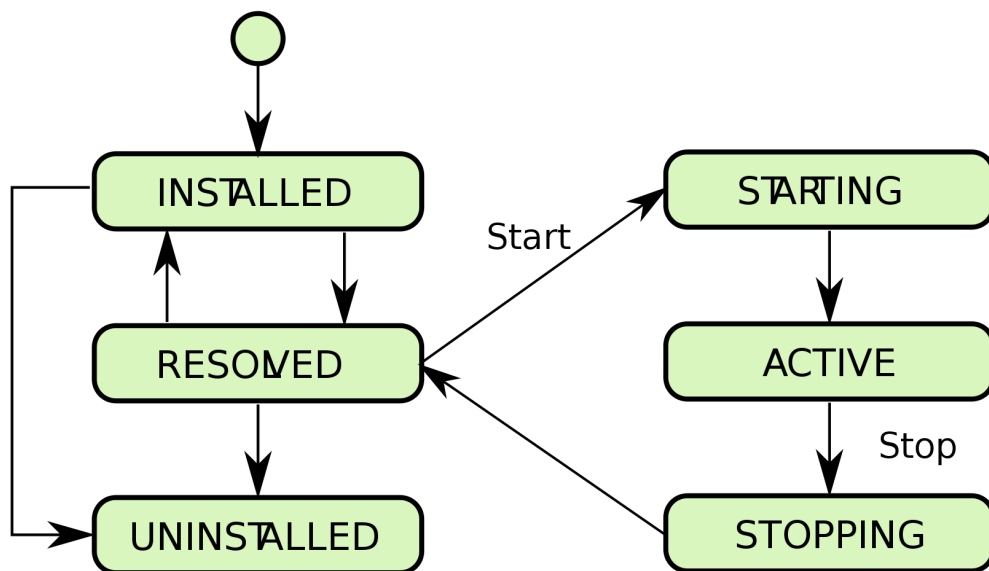


Bundle-Manifest

```
Bundle-Name: Hello World
Bundle-SymbolicName: org.wikipedia.helloworld
Bundle-Description: A Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Bundle-Activator: org.wikipedia.Activator
Export-Package: org.wikipedia.helloworld;version="1.0.0"
Import-Package: org.osgi.framework;version="1.3.0"
```

Bundle-Name	Defines a human-readable name for this bundle, Simply assigns a short name to the bundle.
Bundle-SymbolicName	The only required header, this entry specifies a unique identifier for a bundle, based on the reverse domain name convention (used also by the java packages).
Bundle-Description	A description of the bundle's functionality.
Bundle-ManifestVersion	This little known header indicates the OSGi specification to use for reading this bundle.
Bundle-Version	Designates a version number to the bundle.
Bundle-Activator	Indicates the class name to be invoked once a bundle is activated.
Export-Package	Expresses what Java packages contained in a bundle will be made available to the outside world.
Import-Package	Indicates what Java packages will be required from the outside world, in order to fulfill the dependencies needed in a bundle.

Lebenszyklus eines Bundles



INSTALLED	The bundle has been successfully installed.
RESOLVED	All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.
STARTING	The bundle is being started, the BundleActivator.start method will be called, and this method has not yet returned. When the bundle has an activation policy, the bundle will remain in the STARTING state until the bundle is activated according to its activation policy.
ACTIVE	The bundle has been successfully activated and is running; its Bundle Activator start method has been called and returned.
STOPPING	The bundle is being stopped. The BundleActivator.stop method has been called but the stop method has not yet returned.
UNINSTALLED	The bundle has been uninstalled. It cannot move into another state.

Bildquelle: Faisal Akeel

Open-Source-OSGi-Container

- Equinox
- Knopflerfish
- Apache Felix

Equinox is the reference implementation for the framework portion of the OSGi Service Platform Release 4. It is the modular Java runtime at the heart of the Eclipse IDE, and implements all of the mandatory and most of the optional features of the OSGi R4 specification.

Knopflerfish is an open source implementation of the OSGi R3 and OSGi R4 specifications. Knopflerfish 2 implements all the mandatory features and some of the optional features defined in the R4 specification.

Apache Felix is the open source OSGi container from the Apache Software Foundation. At the time of writing this container is not fully compliant with the OSGi R4 specification.

Quelle: Wikipedia

Eclipse-Demo: Einfaches Bundle anlegen

- ① Neues Plug-in-Projekt “File → New → Project”
- ② Selektiere “Plug-in Project” und “Next”.
- ③ Eingabe:
 - Project Name: com.javaworld.sample.HelloWorld
 - Target Platform: OSGi framework → Equinox
- ④ Weiter mit “Next”.
- ⑤ Selektiere Voreinstellungen im Plug-in-Context-Dialog und “Next”
- ⑥ Templates-Dialog “Hello OSGi Bundle” und “Finish”
- ⑦ Manifest anschauen.
- ⑧ Quellcode von Activator anschauen

Eclipse-Demo: Einfaches Bundle starten

- 1 Framework testweise starten: Sektion "Testing", Link "Launch the framework"
- 2 Status des Bundles anschauen: "ss Hello" in OSGI-Konsole eingeben.
- 3 Bundle anhalten: "stop <nummer>"
- 4 Bundle starten: "start <nummer>"
- 5 Quelltext von Activator verändern: Ausgabe verändern.
- 6 Bundle aktivieren: "update <nummer>"

413 / 504

Eclipse-Demo: Export

- 1 Neues Plug-in-Projekt `com.javaworld.sample.HelloService` anlegen (wie oben)
- 2 Im Projekt neues Interface *HelloService* im Package *com.javaworld.sample.service*:

```
package com.javaworld.sample.service;  
public interface HelloService {  
    public String sayHello();  
}
```

- 3 Neue Klasse *HelloServiceImpl* im Package *com.javaworld.sample.service.impl*:

```
package com.javaworld.sample.service.impl;  
import com.javaworld.sample.service.HelloService;  
  
public class HelloServiceImpl implements HelloService {  
    @Override  
    public String sayHello() {  
        return "at_your_service";  
    }  
}
```

- 4 In MANIFEST.MF im Reiter *Runtime* als *Exported Packages*

414 / 504

Eclipse-Demo: Import

- 1 In MANIFEST.MF von Plug-In HelloWorld im Reiter *Required Plug-ins* *com.javaworld.sample.HelloService* hinzufügen
- 2 Editiere *com.javaworld.sample.helloworld.Activator.java*:
HelloService-Interface ist sichtbar, aber nicht
HelloServiceImpl-Klasse:

```
public void callService(HelloService service) {  
    System.out.println(" HelloWorld:_" + service.sayHello());  
}
```

415 / 504

Eclipse-Demo: Service-Provider

Im Plug-in-Projekt *com.javaworld.sample.HelloService* die Klasse *Activator* wie folgt abändern:

```
package com.javaworld.sample.helloservice;  
import org.osgi.framework.BundleActivator;  
  
public class Activator implements BundleActivator {  
  
    ServiceRegistration helloServiceRegistration;  
    public void start(BundleContext context) throws Exception {  
        System.out.println(" Service:_Hello_World!!");  
        HelloService helloService = new HelloServiceImpl();  
        helloServiceRegistration  
            = context.registerService(HelloService.class.getName(),  
                                     helloService, null);  
    }  
    public void stop(BundleContext context) throws Exception {  
        System.out.println(" Service:_Goodbye_World!!");  
        helloServiceRegistration.unregister();  
    }  
}
```

416 / 504

In OSGi, a source bundle registers a POJO (you don't have to implement any interfaces or extend from any superclass) with the OSGi container as a service under one or more interfaces. The target bundle can then ask the OSGi container for services registered under a particular interface. Once the service is found, the target bundle binds with it and can start calling its methods.

Quelle: <http://www.javaworld.com/javaworld/jw-03-2008/jw-03-osgi1.html?page=4>

Eclipse-Demo: Service-Consumer

Im Plug-in-Projekt *com.javaworld.sample>HelloWord* die Klasse *Activator* wie folgt abändern:

```
public class Activator implements BundleActivator {

    ServiceReference helloServiceReference;

    public void start(BundleContext context) throws Exception {
        System.out.println("Consumer: _Hello _World!!");
        helloServiceReference
            = context.getServiceReference(HelloService.class.getName());
        HelloService helloService
            = (HelloService)context.getService(helloServiceReference);
        System.out.println(helloService.sayHello());
    }
}
```

Eclipse-Demo: Service-Consumer (Forts.)

```
public void stop(BundleContext context) throws Exception {  
    System.out.println("Consumer: _Goodbye_World!");  
    context.ungetService(helloServiceReference);  
}  
}
```

418 / 504

Eclipse-Demo: Start von Service-Provider/Consumer

- 1 Im Manifest von *com.javaworld.sample.HelloService* "Launch the framework" auswählen.
- 2 "ss Hello"
- 3 "update <nummer>", wobei *nummer* die Id des Consumer-Plug-in *HelloWorld* ist

419 / 504

Wiederholungs- und Vertiefungsfragen

- Erläutern Sie die Aspekte eines Komponentenmodells anhand von OSGI?
- Welche Art der Schnittstelleninformation ist in einer Manifest-Datei von OSGI enthalten? Welche Informationen der Schnittstelle sind hingegen nur im Programmcode erkennbar?
- Beschreiben Sie den Lebenszyklus eines Bundles in OSGI.
- Wie können Services eines Bundles von anderen Bundles aufgerufen werden?

420 / 504

Software-Produktlinien

Software-Produktlinien

Lernziele

Software-Wiederverwendung

Erfolgsgeschichten

Definition

Übersicht

Kostenaspekte

Practice Areas

Entwicklung der Core-Assets

Produktentwicklung

Essentielle Aktivitäten

Einführung von Produktlinien

Implementierungsstrategien

Schwierigkeiten

Wiederholungsfragen

421 / 504

Lernziele

Software-Produktlinien

- Definition und Bedeutung
- Vor- und Nachteile
- Technische Aspekte
- Organisatorische Aspekte

N.B.: Basiert auf Folien von Linda Northrop

`http:`

`//www.sei.cmu.edu/productlines/presentations.html`

422 / 504

Software-Wiederverwendung

1960: Unterprogramme

1970: Module

1980: Objekte

1990: Komponenten

→ opportunistische Wiederverwendung im Kleinen; hat nicht den erwarteten Erfolg gebracht

Software-Produktlinien: geplante Wiederverwendung auf allen Ebene für Familien ähnlicher Systeme

423 / 504

Wiederverwendbares

- Komponenten
- Software-Dokumentation
- Architektur
- Tests (unter anderem Integrations-, Leistungs- und Komponententests)
- Anforderungsspezifikation
- Entwicklungsprozess, Methoden und Werkzeuge
- Budget-/Zeit- und Arbeitspläne
- Handbücher
- Entwickler

424 / 504

Erfolgsgeschichten

CelsiusTech: Familie von 55 Schiffssystemen

- Integrationstest of 1-1,5 Millionen SLOC benötigt 1-2 Leute
- Rehosting auf neue Plattform/Betriebssystem benötigt 3 Monate
- Kosten- und Zeitplan werden eingehalten
- Systemattribute (wie Performanz) können vorausgesagt werden
- hohe Kundenzufriedenheit
- Hardware-/Software-Kostenverhältnis veränderte sich von 35:65 zu 80:20

425 / 504

Erfolgsgeschichten

Nokia: Produktlinie mit 25-30 neuen Produkten pro Jahr
Produktübergreifend gibt es

- unterschiedliche Anzahlen von Tasten
- unterschiedliche Display-Größen
- andere unterschiedliche Produktfunktionen
- 58 verschiedene unterstützte Sprachen
- 130 bediente Länder
- Kompatibilität mit früheren Produkten
- konfigurierbare Produktfunktionen
- Änderung der Geräte nach Auslieferung

426 / 504

Software-Produktlinie

Definition

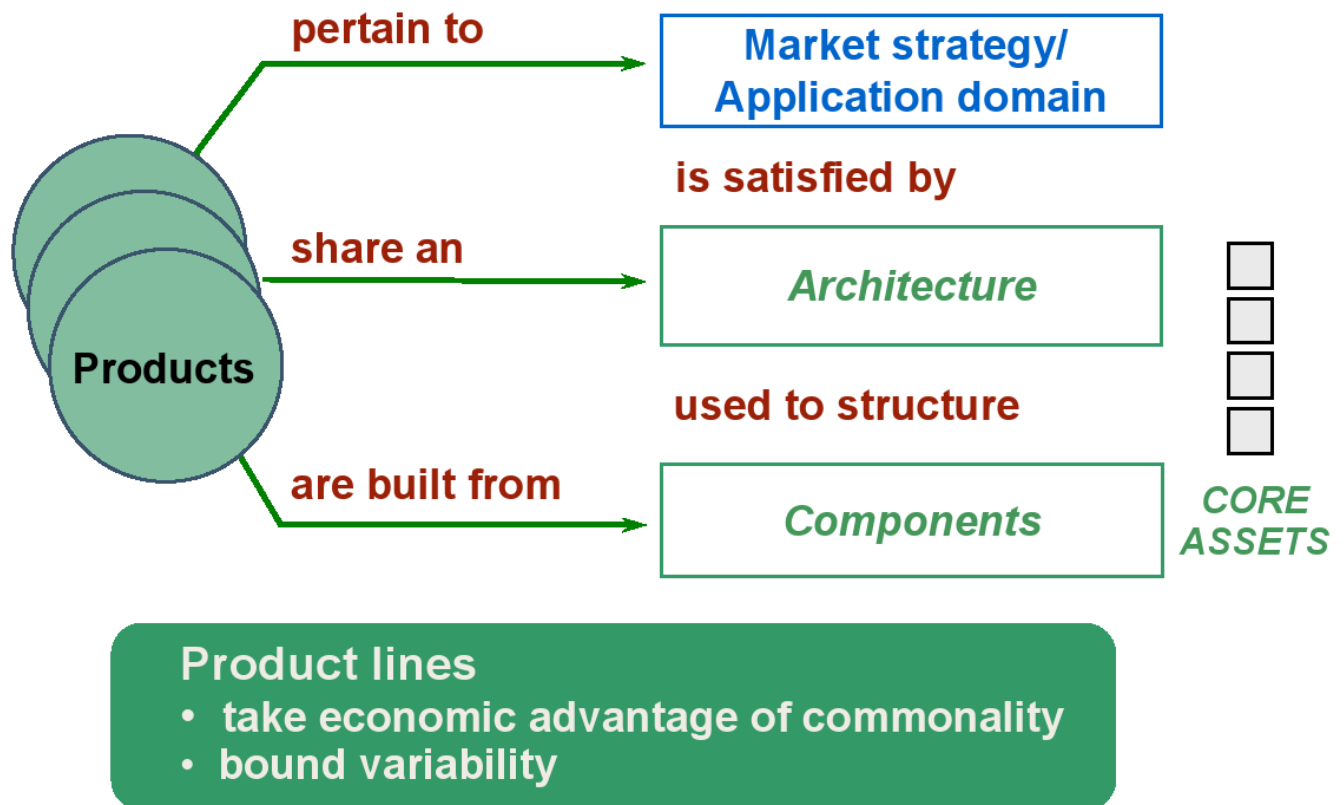
A **software product line** is a set of software-intensive systems

- sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission
- and that are developed from a common set of core assets in a prescribed way.

– Clements und Northrop (2001)

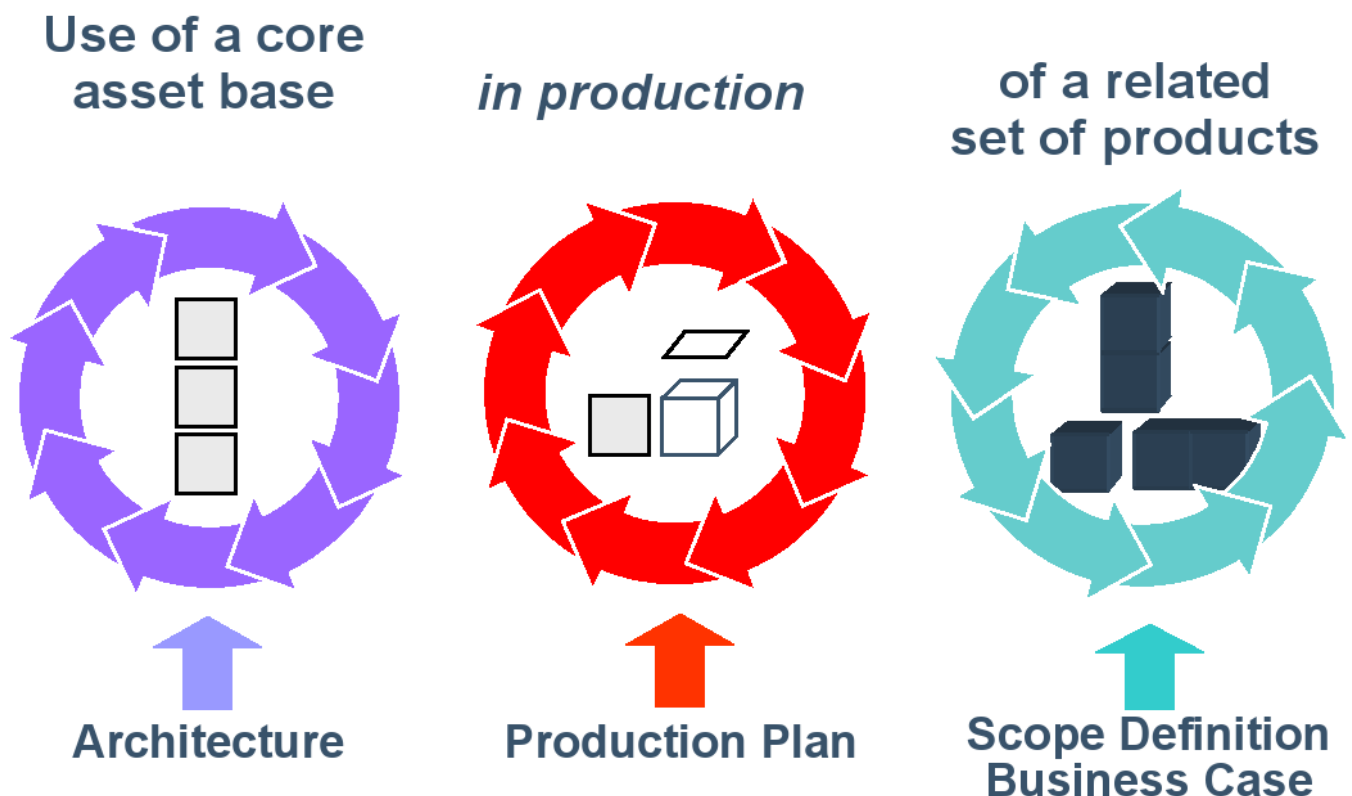
427 / 504

Übersicht über Produktlinien



Quelle: Linda Northrop, SEI 428 / 504

Schlüsselkonzepte



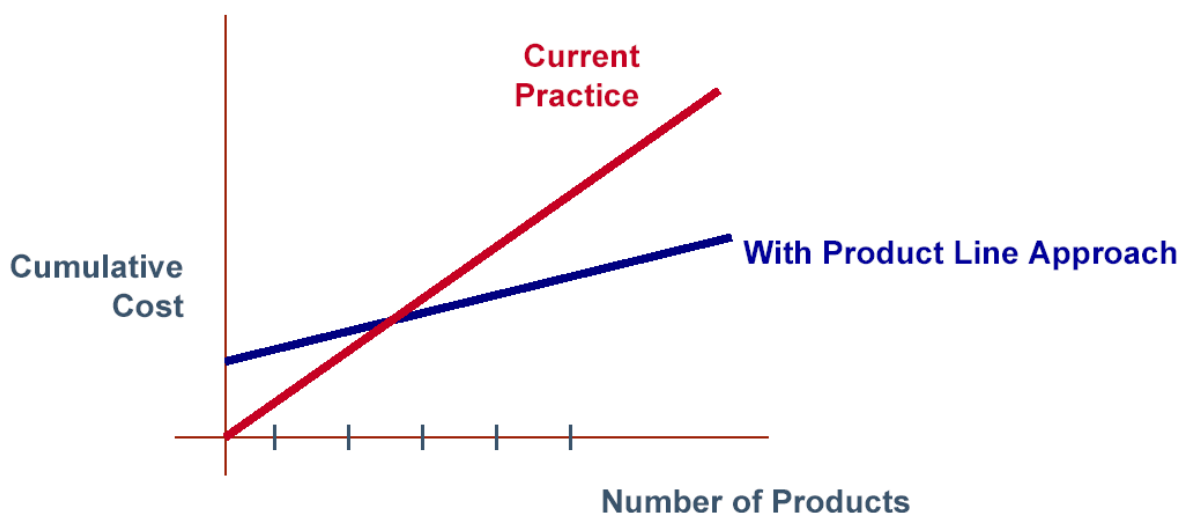
Quelle: Linda Northrop, SEI 429 / 504

Kostenaspekte einer Software-Produktlinie

- Marktanalyse: muss eine Familie von Produkten betrachten
- Projektplan: muss generisch oder erweiterbar sein, um Variationen zu erlauben
- Architektur: muss Variation unterstützen
- Software-Komponenten: müssen generischer sein, ohne an Performanz einzubüßen; müssen Variation unterstützen
- Testpläne/-fälle/-daten: müssen Variationen und mehrere Instanzen einer Produktlinie berücksichtigen
- Entwickler: benötigen Training in den Assets und Verfahren der Produktlinie

430 / 504

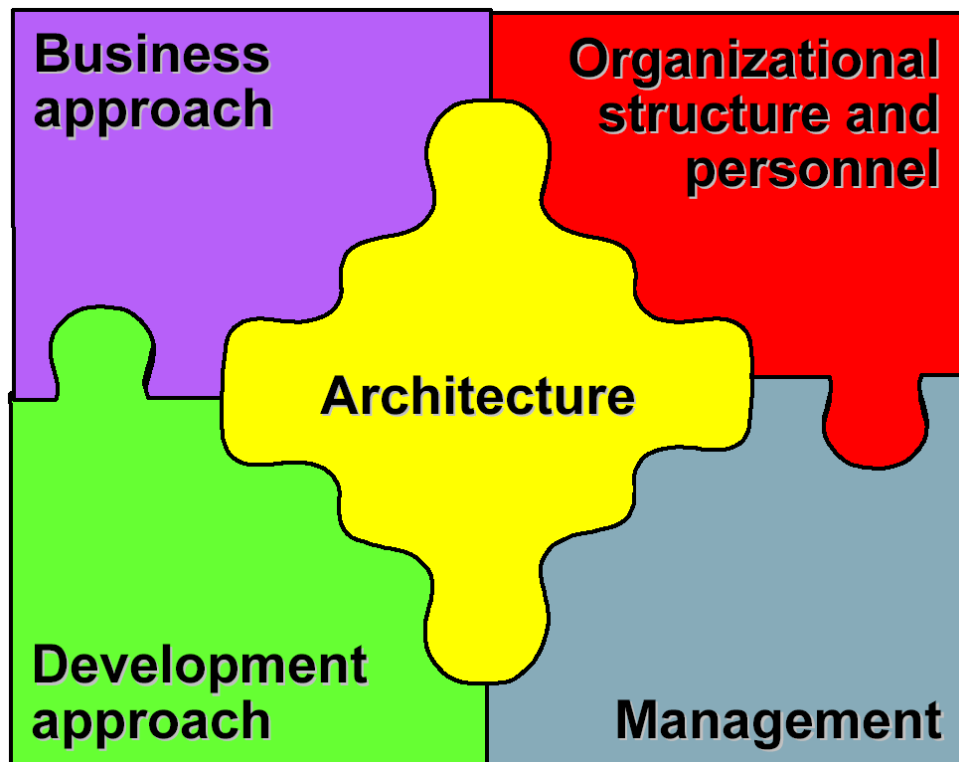
Return-on-Investment



Quelle: Weiss und Lai, 1999.

431 / 504

Zusammenspielende Komponenten



Quelle: Linda Northrop, SEI

432 / 504

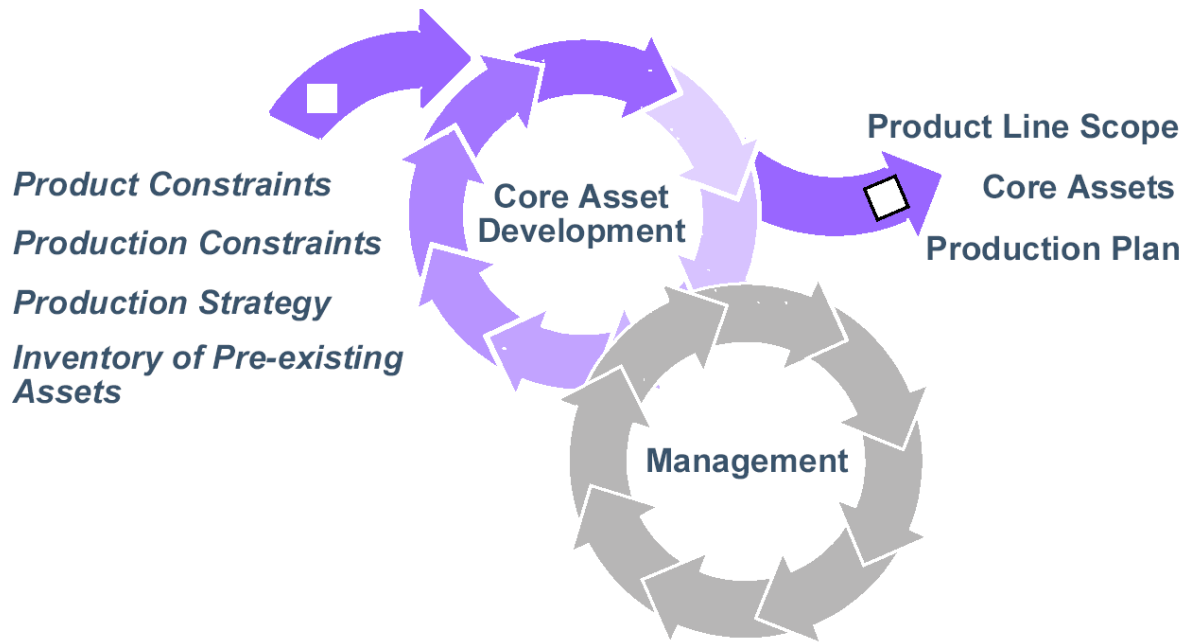
Practice Areas



Quelle: Linda Northrop, SEI

433 / 504

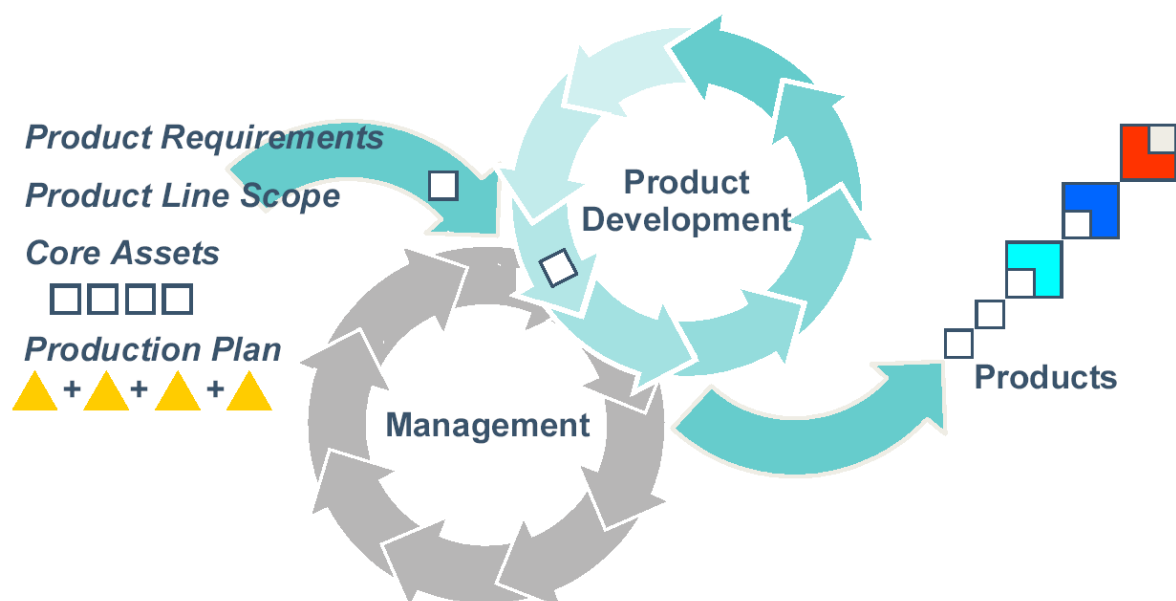
Entwicklung der Core-Assets



Quelle: Linda Northrop, SEI

434 / 504

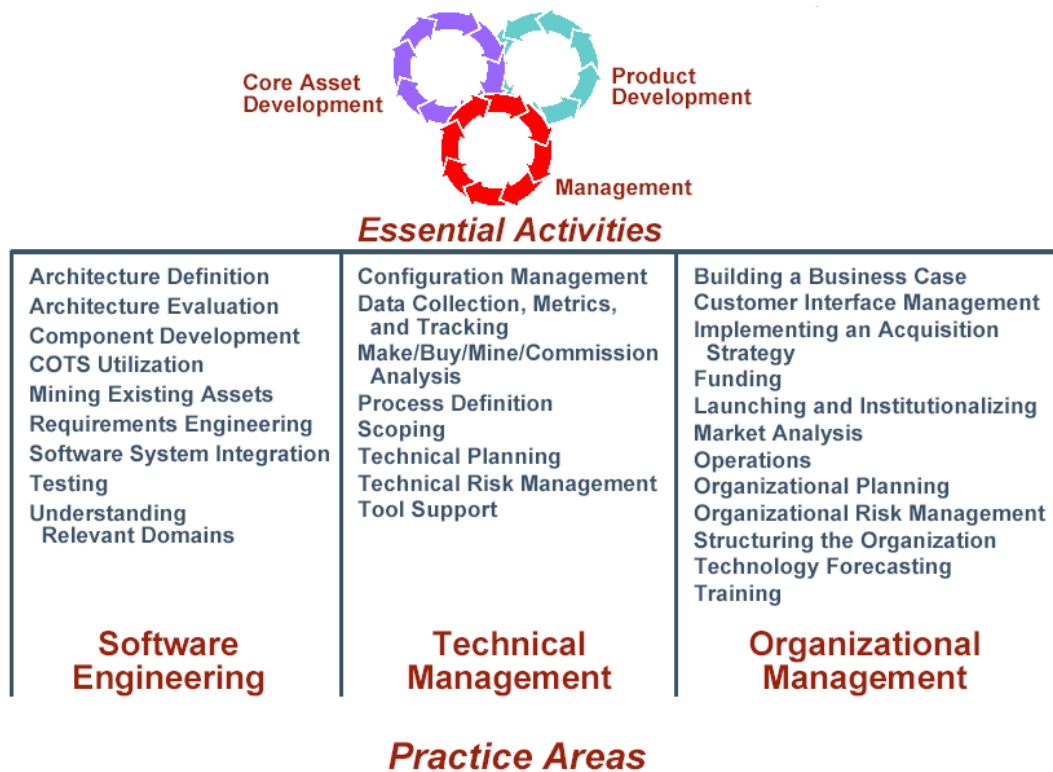
Produktentwicklung



Quelle: Linda Northrop, SEI

435 / 504

Essentielle Aktivitäten



Quelle: Linda Northrop, SEI

436 / 504

Einführung von Produktlinien I

Proaktiv:

- definiere zuerst Scope: was gehört zur Produktlinie?
 - Scope leitet die weitere Entwicklung
 - Entwickle zuerst Core-Assets
- + Produkte können rasch entwickelt werden, sobald die Produktlinie steht
- hohe Vorausleistung und Vorhersagefähigkeit verlangt

Einführung von Produktlinien II

Reaktiv:

- Beginne mit einem oder mehreren Produkten
- Extrahiere daraus Core-Assets für die Produktlinie
- Scope entwickelt sich dabei stetig
- + niedrige Einstiegskosten
- + größerer Einfluss von Erfahrung
- Architektur könnte suboptimal sein, wird schrittweise weiterentwickelt
- Restrukturierungsaufwand notwendig

438 / 504

Einführung von Produktlinien III

Inkrementell (sowohl bei reaktiver als auch proaktiver Entwicklung möglich):

- schrittweise Entwicklung der Core-Assets mit initialer Planung der Produktlinie:
- entwickle Teile der Core-Asset-Base einschließlich Architektur und Komponenten
- entwickle ein oder mehrere Produkte
- entwickle weitere Core-Assets
- entwickle weitere Produkte
- entwickle Core-Asset-Base weiter
- ...

439 / 504

Bindung

Produktlinien ...

- haben Gemeinsamkeiten
- und definierte Unterschiede: Variabilitäten

Produkt wird aus Core-Assets zusammengebaut.
Variabilitäten werden festgelegt.

Bindungszeitpunkt der Variabilitäten

- zur Übersetzungszeit
- zur Bindezeit
- zur Laufzeit

440 / 504

Architekturmechanismen für Variabilitäten

Kombination, Ersetzung und Auslass von Komponenten (auch zur Laufzeit)

Frontend {C, C++, Java}	Middle End {ME}	Backend {i386, Motorola 68000}
C	ME	i386
C	ME	Motorola 68000
C++	ME	i386
C++	ME	Motorola 68000
Java	ME	i386
Java	ME	Motorola 68000

441 / 504

Architekturmechanismen für Variabilitäten

Parametrisierung (einschließlich Makros und Templates)

```
1 generic
2   type My_Type is private;
3   with procedure Foo (M : My_Type);
4   procedure Apply;
5
6   procedure Apply is
7     X : My_Type;
8   begin
9
10    ...
11    Foo (X);
12    ...
13 end Apply;
```

442 / 504

Architekturmechanismen für Variabilitäten

Parametrisierung (einschließlich Makros und Templates)

```
1 typedef (*FP)(int);
2 void Apply (FP fp) {
3   ...
4   fp (X);
5   ...
6 }
```

443 / 504

Architekturmechanismen für Variabilitäten

Selektion verschiedener Implementierungen zur Übersetzungszeit
(z.B. `#ifdef` oder `Makefile`)

```
1 #ifdef Kunde1
2 #define My_Type int
3 void Foo (My_Type M) {...}
4 #else
5 typedef struct mystruct {...} mystruct;
6 #define My_Type mystruct
7 void Foo (My_Type M) {...}
8 #endif
9 void Apply () {
10     My_Type X;
11     ...
12     Foo (X);
13     ...
14 }
```

444 / 504

Architekturmechanismen für Variabilitäten

Entwurfsmuster mit OO-Techniken

- Strategy
- Factory
- Template Method
- ...

445 / 504

Architekturmechanismen für Variabilitäten

Generierung (z.B. Yacc: Grammatik → Code) und aspektorientierte Programmierung

```
1 aspect SetsInRotateCounting {
2     int rotateCount = 0;
3
4     before(): call(void Line.rotate(double)) {
5         rotateCount++;
6     }
7 }
```

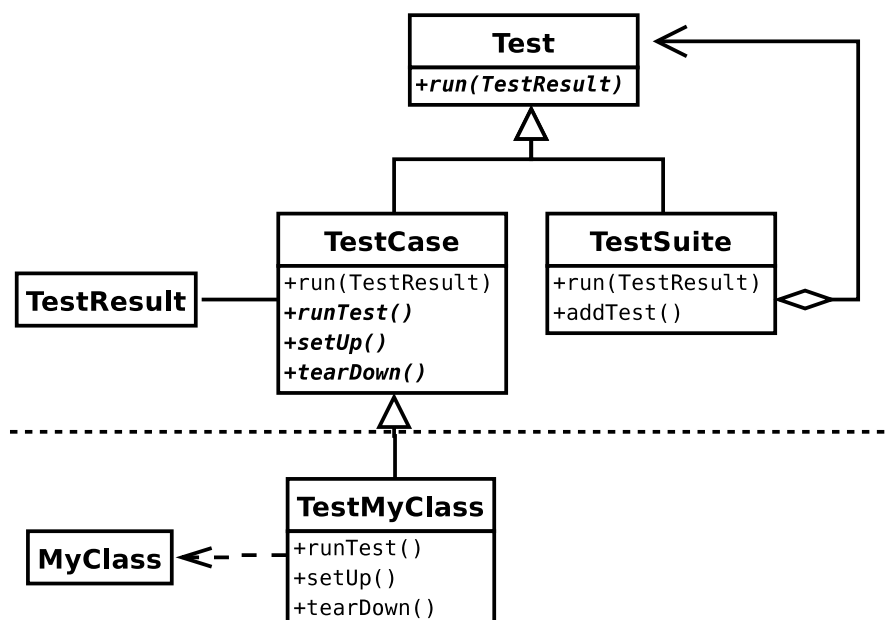
- Wie oft wird Methode `Line.rotate` aufgerufen?

446 / 504

Architekturmechanismen für Variabilitäten

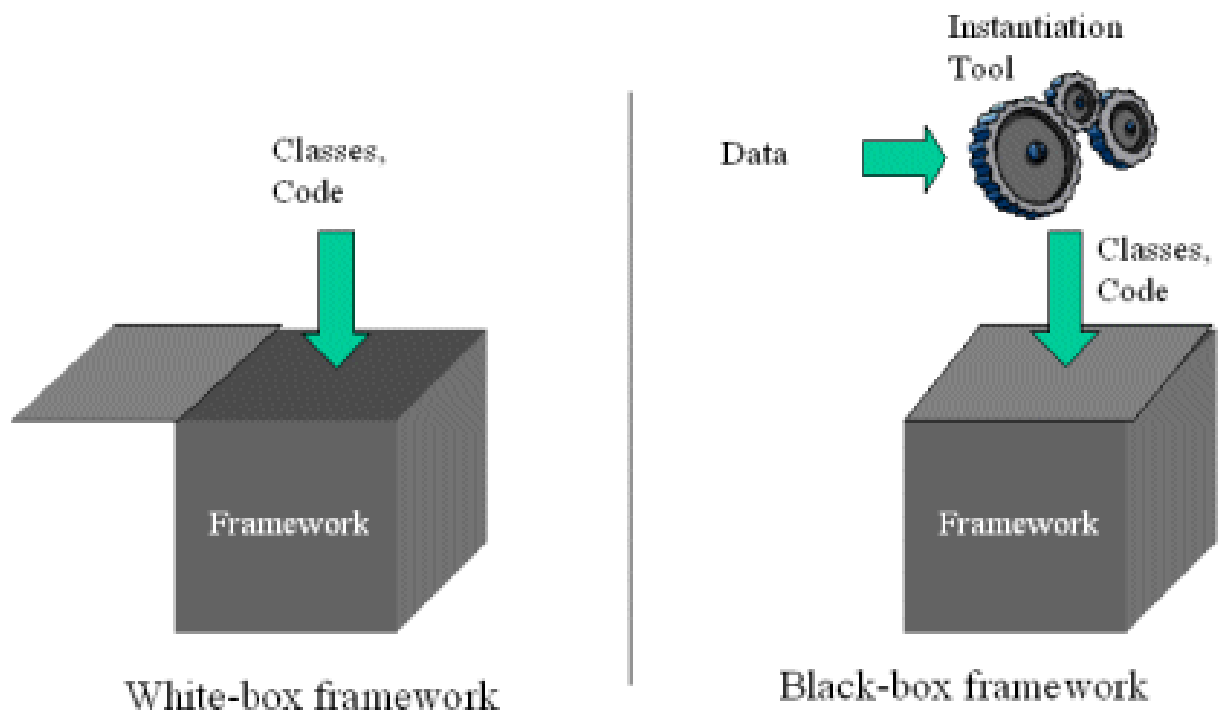
Definition

Anwendungsrahmenwerke (Frameworks): A framework is a set of classes that embodies an abstract design for solutions to a family of related problems. Johnson und Foote (1988)



447 / 504

Anwendungsrahmenwerke (Frameworks)



448 / 504

Schwierigkeiten bei Produktlinien

- Änderung der Organisation (Kern-/Produktaufteilung)
- Was gehört zum Kern?
- Änderungen im Kern haben Auswirkungen auf alle Produkte
- Viele Probleme sind noch nicht gelöst:
 - Test
 - Konfigurationsmanagement
 - ...

449 / 504

Wiederholungs- und Vertiefungsfragen

- Erläutern Sie die Ideen von Software-Produktlinien. Was verspricht man sich davon?
- Was sind die Vor- und Nachteile?
- Wie wird die Entwicklung von Software-Produktlinien organisatorisch häufig strukturiert?
- Erläutern Sie einige essentielle Aktivitäten der Softwareentwicklung und ihre Besonderheiten im Kontext von Software-Produktlinien.
- In welcher Weise können Produktlinien eingeführt werden?
- Beschreiben Sie Implementierungsmechanismen für die Variabilität in Software-Produktlinien. Nennen Sie hierbei den jeweiligen Bindungszeitpunkt. Was drückt der Bindungszeitpunkt aus?

450 / 504

Modellgetriebene Softwareentwicklung

Modellgetriebene Softwareentwicklung

Modellgetriebene Entwicklung

Modelle

Metamodelle

Syntax und Semantik von Modellen

Standards

Domänenspezifische Sprachen (DSL)

Werkzeuge für DSLs

Modelltransformationen

Modell-zu-Text-Transformationen

Zusammenfassung

Wiederholungsfragen

451 / 504

Definition

Modellgetriebene Softwareentwicklung (Model-Driven Software Development (MDSD)) bezeichnet

Softwareentwicklungsprozesse, bei denen Modelle im Mittelpunkt stehen und als eigenständige Entwicklungsartefakte genutzt werden (Reussner und Hasselbring 2009).

Modelle sind zentrale Entwicklungsartefakte:

- Kommunikation mit Fachexperten
- Analyse
- Generierung von Code

Ziele: Reduktion der Komplexität durch

- Abstraktion (Reduktion auf das Wesentliche)
- Automatisierung

452 / 504

Diese Folien basieren in großen Teilen auf einem Vortrag von Stefan Gudenkauf, OFFIS Institut für Informatik.

- Handhabung von Komplexität
 - Abstraktion zum Wesentlichen
 - Einbindung von Fachexperten
 - Trennung von Aufgaben und Belangen
- Steigerung der Entwicklungseffizienz
 - Generierung von redundantem Programmcode
 - Wiederverwendung
- Steigerung der Softwarequalität
 - Wohldefinierte Regeln für Modelle
 - Bewährte Transformationen
 - Homogener Programmcode
- Interoperabilität und Portabilität

Merkmale eines Modells nach Stachowiak (1973)

Abbildung

- Repräsentation eines betrachteten Gegenstands
- Übertragbarkeit bestimmter Beobachtungen am Modell auf modellierten Gegenstand

Verkürzung

- Betrachtung der *relevanten* Attribute des betrachteten Systems im Modell für bestimmte Betrachter
- irrelevante Attribute werden nicht repräsentiert

Pragmatismus

- das Modell ist einem bestimmten Zweck zugeordnet
- der Zweck bestimmt Verkürzung und Abbildung

454 / 504

Modell und Metamodell

Definition

Ein **Metamodell** ist das Modell einer Menge von Modellen.

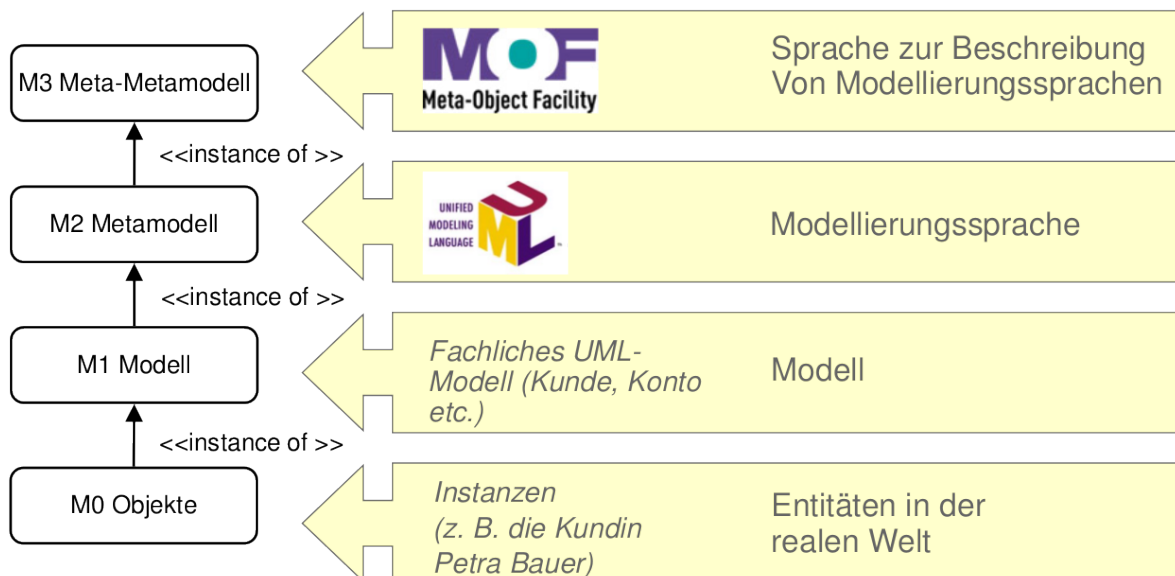
– Favre (2004)

Definition

Ein **Metametamodell** ist das Modell einer Menge von Metamodellen.

455 / 504

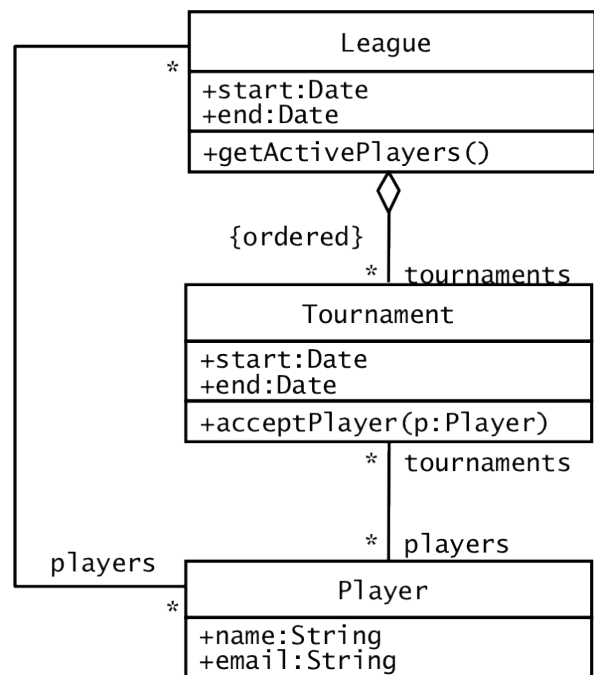
Ein Modell ist selbst ein Betrachtungsgegenstand und kann modelliert werden. Ein Metamodell ist selbst wieder ein Modell und wird durch ein Metamodell beschrieben: ein Metametamodell.



Syntax und Semantik von Modellen (Stahl u. a. 2007)

Konkrete Syntax

- Definiert die konkrete Darstellung von Modellen
- Regeln für die Abbildung auf die abstrakte Syntax



457 / 504

Konkrete Syntax: Eine Klasse wird als Rechteck gezeichnet. ... Bildquelle: OCL Tutorial von Mazeiar Salehie

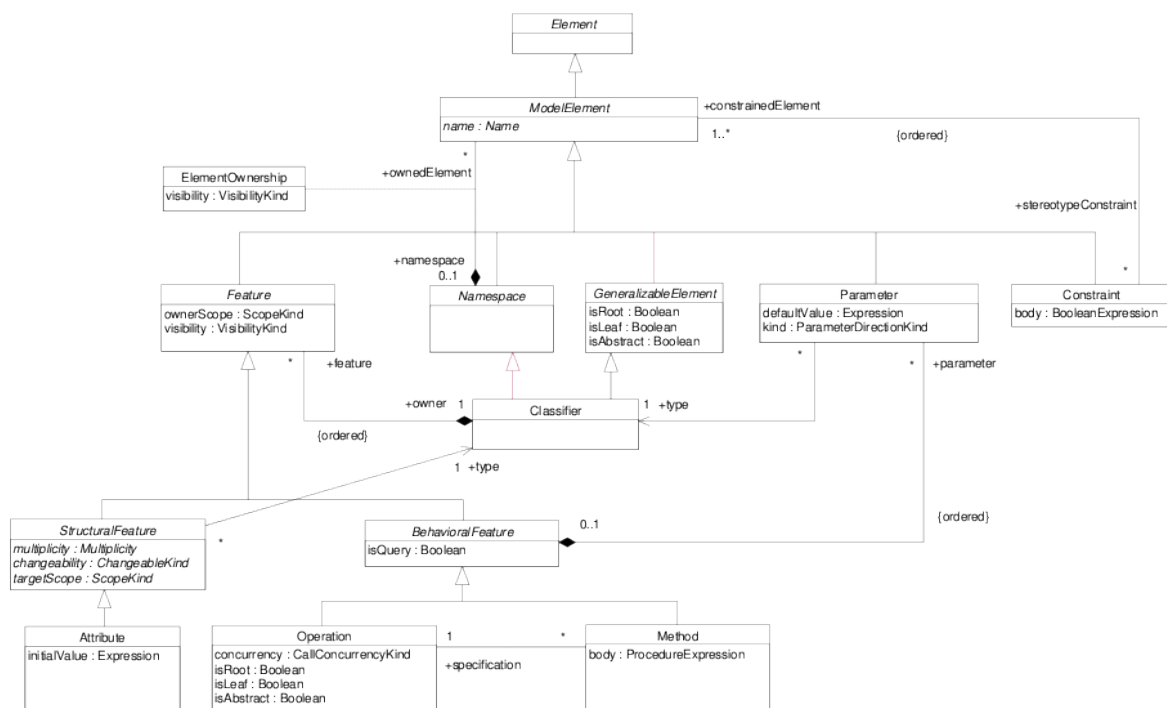
<http://www.stargroup.uwaterloo.ca/~ltahvild/courses/ECE493-T5/tutorials/Tutorial-Feb16-OCL.pdf>

Abstrakte Syntax

- Definiert den Aufbau korrekter Instanzen
- Elemente und ihre Beziehungen

Abstrakte Syntax: Eine Klasse enthält Attribute und Methoden

Abstrakte Syntax von UML-Klassendiagrammen (Ausschnitt)



Class leitet von *Classifier* ab.

Syntax und Semantik nach Stahl u. a. (2007)

Statische Semantik

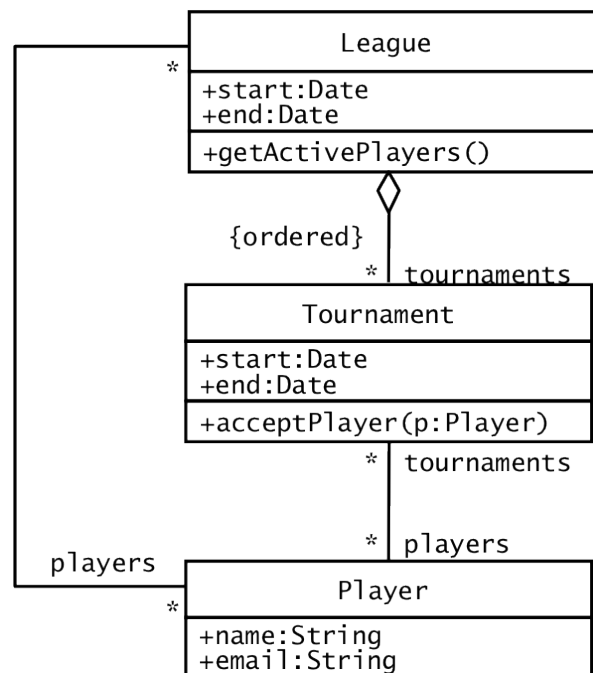
- Schränkt die abstrakte Syntax ein

OCL-Beispiel:

context Tournament

inv: end - start $\leq$

Calendar.WEEK



460 / 504

Statische Semantik: Der Name einer Klasse muss eindeutig sein. Kann z.B. mit Object Constraint Language (OCL)

ausgedrückt werden. Bildquelle und OCL-Beispiel: OCL Tutorial von Mazeiar Salehie

<http://www.stargroup.uwaterloo.ca/~ltahvild/courses/ECE493-T5/tutorials/Tutorial-Feb16-OCL.pdf>

„Dynamische“ Semantik

- Bedeutung bzw. Interpretation der abstrakten Syntax
- z.B. formalisiert als Abbildung der abstrakten Syntax auf ein mathematisches Modell (denotionale Semantik)
- z.B. formalisiert als Abbildung auf ausführbares Modell (z.B. Code) (operationale Semantik)

„Dynamische“ Semantik: Jede Instanz der Klasse hat alle Attribute ihrer Klasse. Methoden der Klasse können diese lesen und schreiben.

Standards der modellgetriebenen Entwicklung

- Model Driven Architecture (MDA)
- UML
- MOF

462 / 504

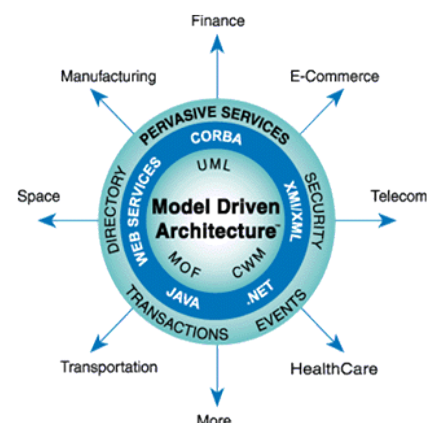
Standard: Model Driven Architecture (MDA)

MDA: Ansatz der Object Management Group (OMG) zur modellgetriebenen Entwicklung mit den Zielen:

- Interoperabilität
- Portabilität
- Wiederverwendbarkeit

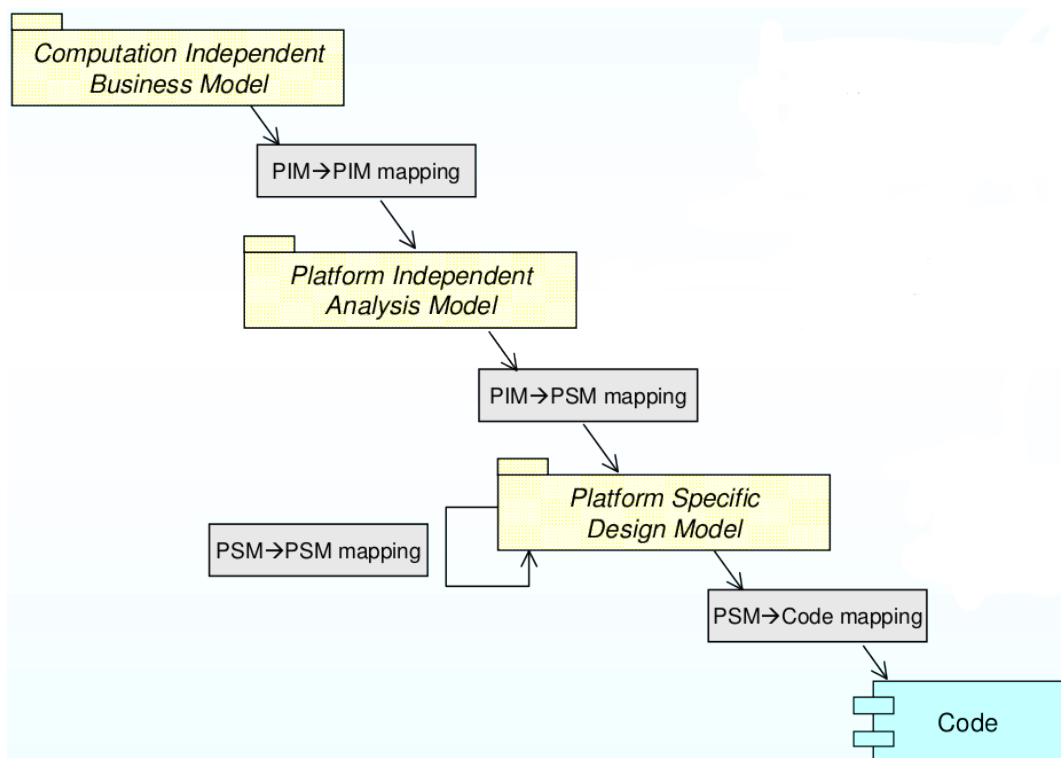
Verbindet verschiedene OMG-Standards

- Meta Object Facility (MOF)
- Unified Modeling Language (UML)
- Common Warehouse Model (CWM)
- Query/Views/Transformations (QVT)
- XML Metadata Interchange (XMI)



463 / 504

Model-Driven Architecture



Computation Independent Model (CIM):

- Modell der Domäne
- Enthält die Anforderungen an das zu entwickelnde System

Platform Independent Model (PIM)

- Modell der Implementierung unabhängig von der Zielplattform
- Grundlage für die Entwicklung eines Systems auf verschiedenen Zielplattformen

Platform Specific Model (PSM)

- Ergänzt das PIM mit Details zu einer bestimmten Plattform
- Evtl. zusätzlich Platform Model zwischen PIM und PSM

Code

- Ausführbarer Quellcode

Standard: UML 2.x

UML-Infrastructure

- Grundlagen der abstrakten Syntax
- z.B. Klassen, Assoziationen, Multiplizitäten

UML-Superstructure

- Erweiterungen der UML-Infrastructure um Elemente für spezielle Modellierungsaufgaben
- z.B. Komponenten, Anwendungsfälle, Aktivitäten

Object Constraint Language (OCL)

- Beschreibung der statischen Semantik
- Invarianten, Vor- und Nachbedingungen etc.

Diagram Interchange (XMI)

- Austausch der grafischen Modellrepräsentation
- z.T. uneinheitlich implementiert

Standard: Meta Object Facility (MOF)

MOF: Standard der Object Management Group zur Metamodellierung

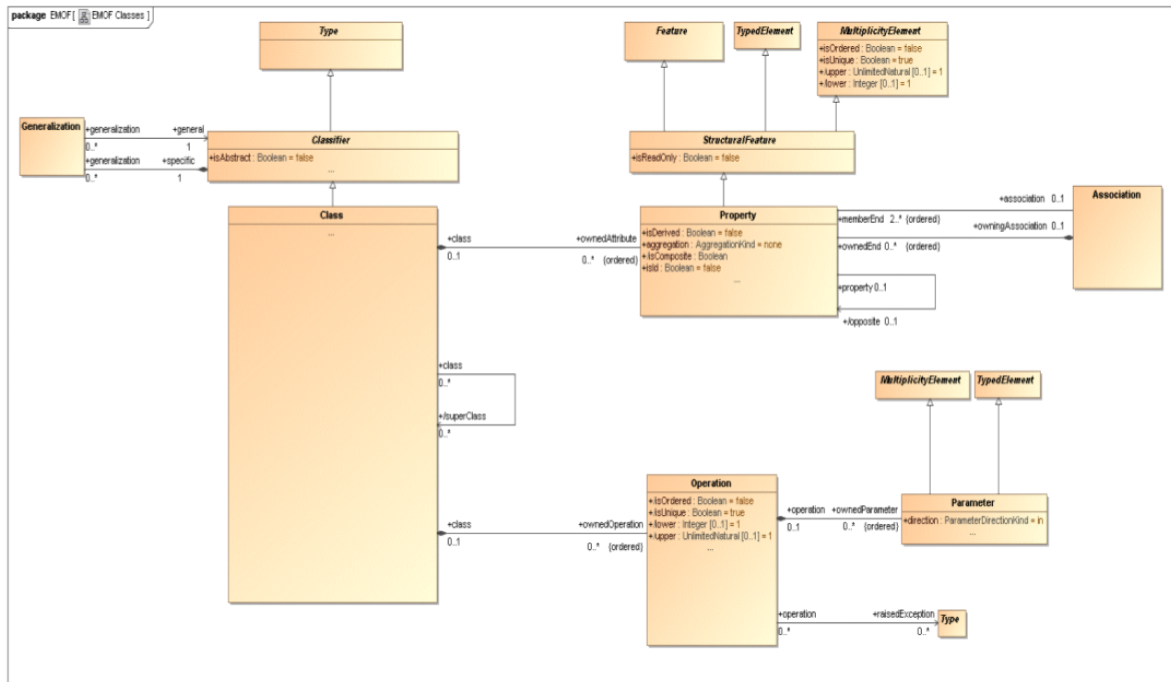
→ basiert auf UML-Klassendiagrammen

Varianten:

- Complete MOF (CMOF):
Gesamter Sprachumfang von MOF
- Essential MOF (EMOF):
 - Minimale Untermenge von MOF, um Metamodelle beschreiben zu können
 - Weitestgehend kompatibel zu *Ecore* aus dem Eclipse Modeling Framework (EMF)



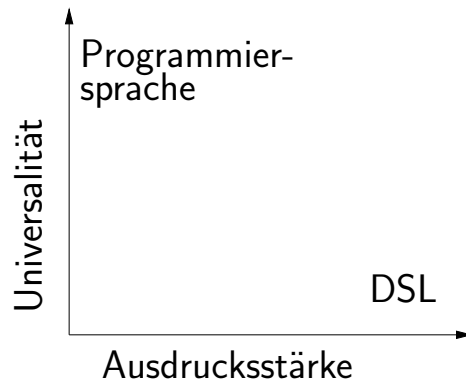
EMOF



Domänenspezifische Sprache

Definition

Domänenspezifische Sprache (Domain-specific Language DSL): Sprachen, die auf eine spezielle Anwendungsdomäne zugeschnitten sind.



468 / 504

They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application. (Mernik u. a. 2005)

DSLs tauschen Universalität (allgemeine Anwendbarkeit) gegen Ausdrucksstärke in einer begrenzten Domäne

Arten von DSLs

Interne DSLs (Language Exploitation): eingebettet in bestehende Sprachen

- Nutzung einer existierenden Wirtssprache (Piggyback)
- Spezialisierung und Erweiterung der Wirtssprache, z.B. UML-Profile als Spezialisierung
- Nutzung bestehender Werkzeuge

Externe DSLs (Language Invention)

- Grammatik (abstrakte Syntax, Metamodell)
- Notation (konkrete Syntax, grafisch oder textuell)
- Benötigen eigene Werkzeuge

469 / 504

Repräsentation von DSLs

Grafische Notation (Rendering)

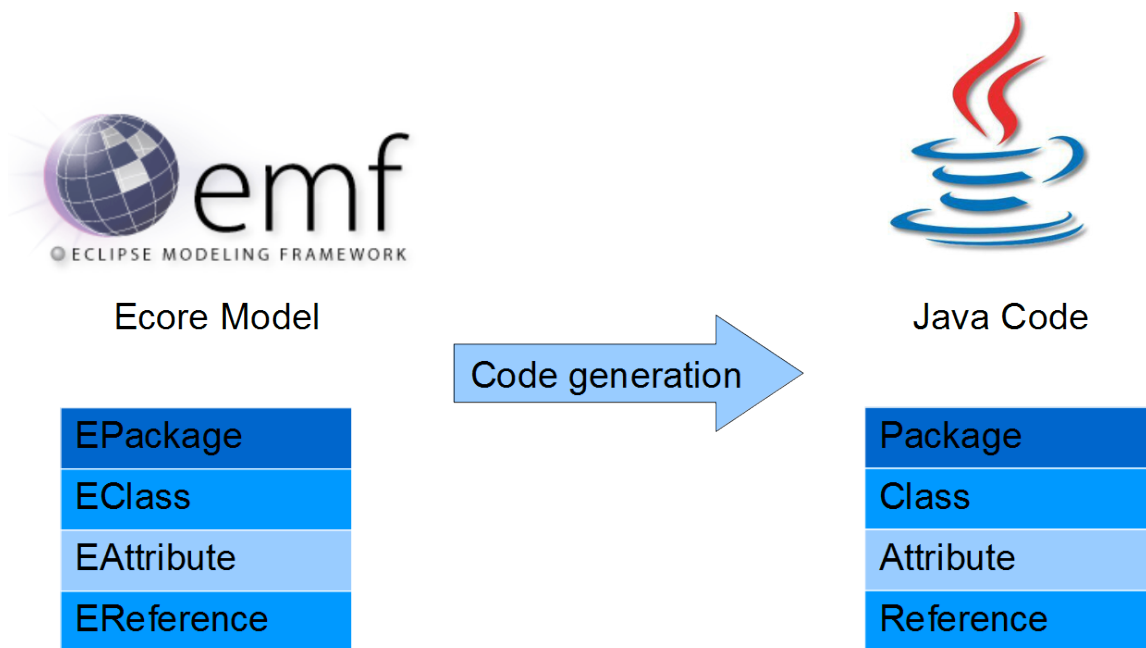
- Hohe Informationsdichte
- Mehrdimensionalität (Kleppe 2008)
- Häufig graphorientiert (Knoten & Kanten)
- Bei größeren Projekten abwägen:
Aufwand Layout > Aufwand Modellierung?

Textuelle Notation (Serialisierung)

- Schnell editierbar, breite Werkzeugunterstützung
- Häufig sehr kompakt und formal
- Häufig blockstrukturiert (Textabschnitte bilden Blöcke)
- Darstellung von Beziehungen zwischen Entitäten schwierig (z.B. Verweise)

470 / 504

Eclipse Modeling Framework (EMF)



471 / 504

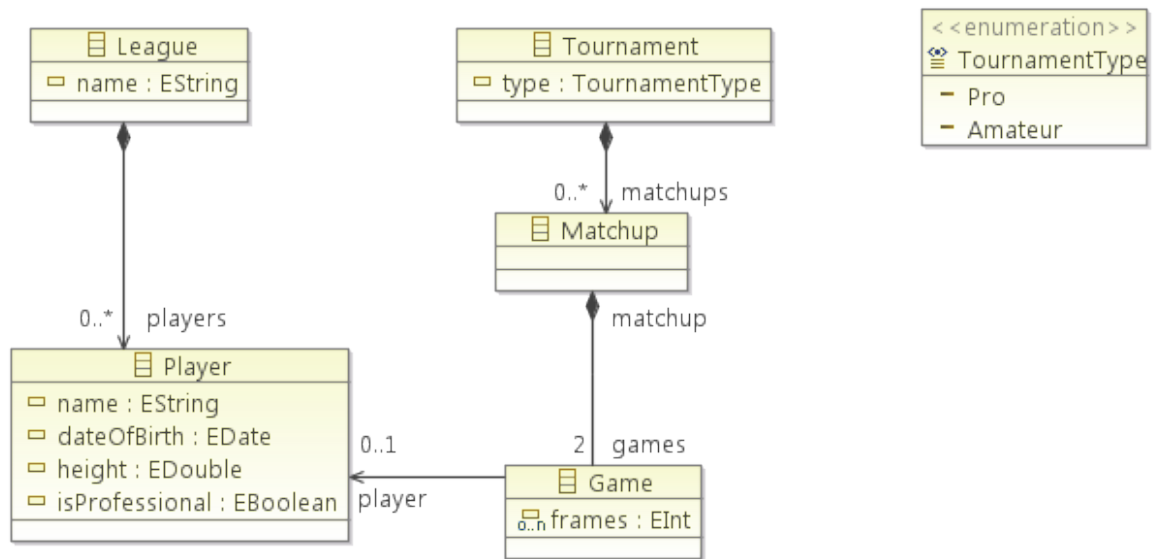
Eclipse-Projekt für die Metamodellierung <http://www.eclipse.org/modeling/emf/> als Teil des Eclipse Modeling Project <http://www.eclipse.org/modeling/>
Ecore

- Kern von EMF
- Implementierung von OMGs Essential MOF (EMOF)
- EClass : represents a class, with zero or more attributes and zero or more references.
- EAttribute : represents an attribute which has a name and a type.
- EReference : represents one end of an association between two classes. It has flag to indicate if it represent a containment and a reference class to which it points.
- EDataType : represents the type of an attribute, e.g. int, float or java.util.Date

Bildquelle: <http://eclipsesource.com/blogs/2011/03/22/>

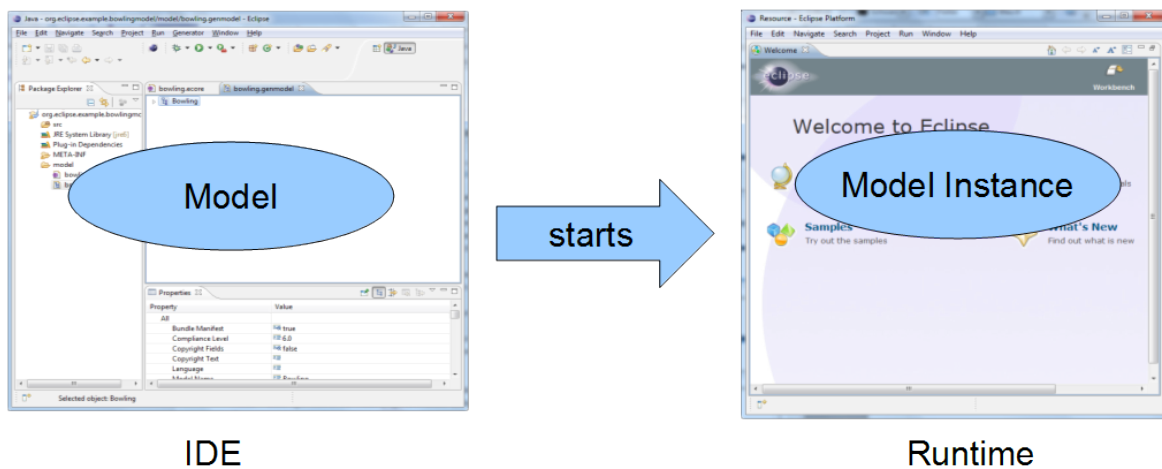
[what-every-eclipse-developer-should-know-about-emf-part-1/](#)

Eclipse Modeling Framework (EMF)



472 / 504

Eclipse Modeling Framework (EMF)



473 / 504

Werkzeuge in EMF

- Generierung von Java-Code aus Ecore-Metamodelle
- Generierung von Java-Code zur Bearbeitung von Metamodellen
- Serialisierung von Metamodellen in XMI (basiert auf XML)
- Baureditor zur direkten Modellierung von Metamodellen und Modellen
- UML-Klassendiagrammartiger grafischer Editor

Graphical Modeling Framework (GMF)

- Eclipse-Projekt zur modellgetriebenen Erstellung grafischer Editoren für Ecore-Modelle
- Teil des Eclipse Modeling Project

Editorbau mit GMF

- Beschreibung von Modellen für verschiedene Aspekte des Editors
- Besonders geeignet für die schnelle Erstellung einfacher grafischer Editoren
- Fortgeschrittene Editoren erfordern Änderungen am Quellcode und an GMF-Templates

Textuelle Modellierung mit Xtext

Xtext – Language Development Framework

- Eclipse-Projekt zur Entwicklung externer textueller DSLs basierend auf EMF
- Teil des Eclipse Modeling Project
- Definition von EBNF-artigen Grammatiken, die abstrakte und konkrete Syntax gleichzeitig darstellen
- Verwendung von Xpand-Templates zur Code-Generierung

```
grammar org.example.domainmodel.Domainmodel
    with org.eclipse.xtext.common.Terminals
```

```
generate domainmodel
    "http://www.example.org/domainmodel/Domainmodel"
```

```
Model:
    greetings+=Greeting*;
```

```
Greeting:
    'Hello' name=ID '!';
```

Modelltransformationen

Definition

Modelltransformationen: Abbildung einer Menge von Modellen auf eine andere Menge von Modellen

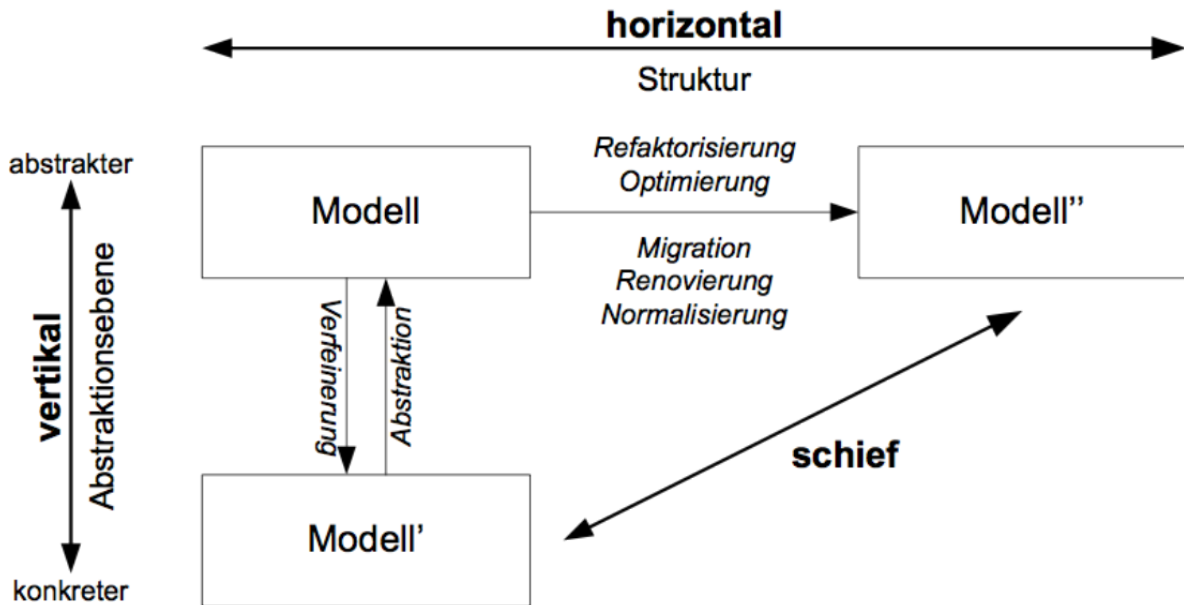
Varianten:

- Modell-zu-Modell-Transformationen (M2M, Mappings)
- Modell-zu-Text-Transformationen (M2T, Templates)

Modelltransformationen werden in der Regel

- auf Metamodellen beschrieben und
- auf Modellinstanzen angewendet

Arten von Modelltransformationen



– Reussner und Hasselbring (2009)

477 / 504

Modell-zu-Modell-Transformationen

Erstellung von Modellen eines anderen Blickwinkels

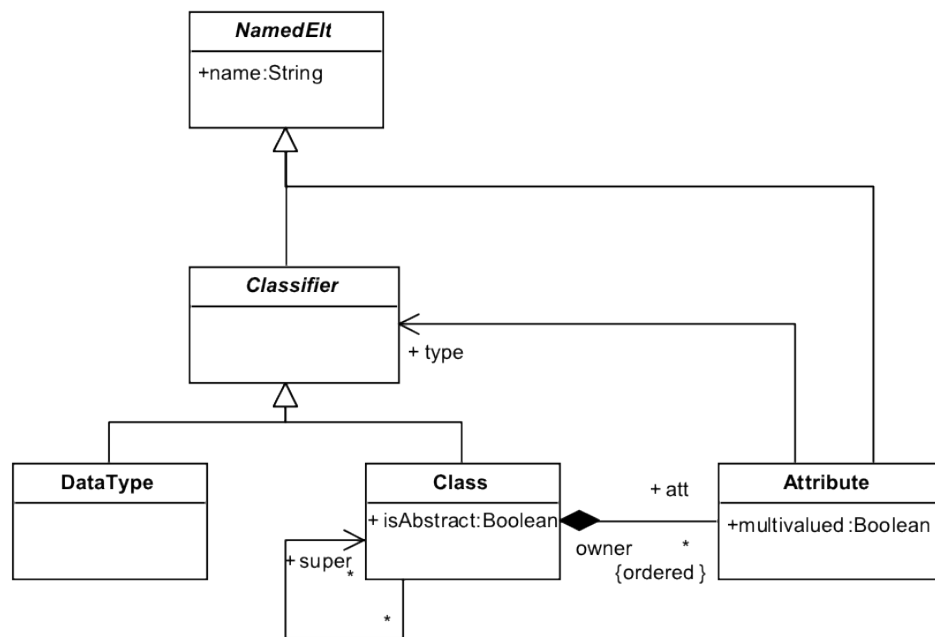
- Überführen von Modellen höherer Abstraktionsebene in niedrigere Abstraktionsebenen
- Verfeinerung von Modellen

M2M-Transformationssprachen

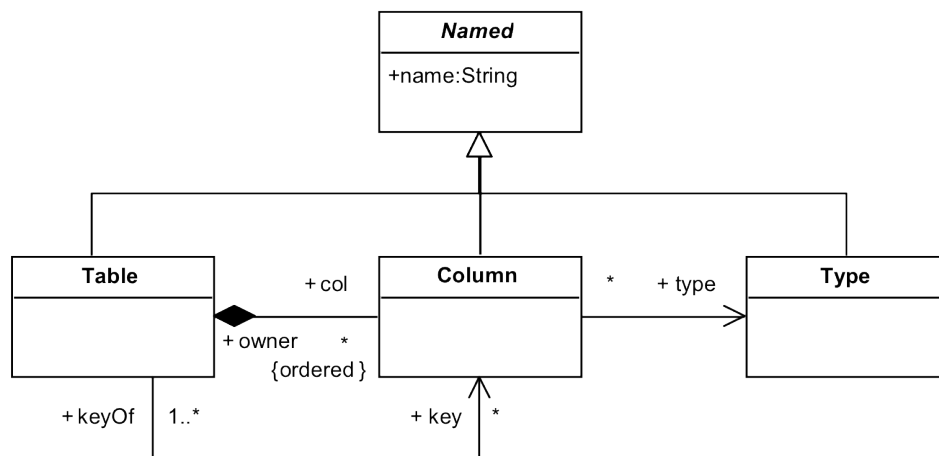
- Query View Transformation (QVT) Operational Mapping
- Query View Transformation (QVT) Relations
- Atlas Transformation Language (ATL)
- Xtend aus dem (Eclipse) Xtext Language Development Framework

478 / 504

ATL-Beispiel: Metamodell für Quelle



ATL-Beispiel: Metamodell für Ziel



480 / 504

ATL-Beispiel: Transformation (Header)

```
module Class2Relational;
create OUT : Relational from IN : Class;
uses strings;
-- inheritance is not supported yet

helper def: objectIdType : Relational!Type =
Class!DataType.allInstances()
->select(e | e.name = 'Integer')->first();
```

481 / 504

ATL-Beispiel: Transformation

```
rule DataType2Type {  
  from  
    dt : Class!DataType  
  to  
    out : Relational!Type (  
      name <- dt.name  
    )  
}
```

482 / 504

For each DataType instance, a Type instance has to be created. Their names have to correspond.

ATL-Beispiel: Transformation

```
rule Class2Table {
from
  c : Class!Class
to
  out : Relational!Table (
    name <- c.name,
    -- Columns are generated from Attributes in another
    -- rule not explicitly called here !
    col <- Sequence {key}->
      union(c.attr->select(e | not e.multiValued)),
    key <- Set {key}
  ),
  key : Relational!Column (
    name <- 'objectId',
    type <- thisModule.objectIdType
  )
}
```

483 / 504

For each Class instance, a Table instance has to be created.

- Their names have to correspond.
- The col reference set has to contain all Columns that have been created for single- valued attributes and also the key described in the following.
- The key reference set has to contain a pointer to the key described in the following.
- An Attribute instance has to be created as key
 - Its name has to be set to 'objectId'
 - Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.

ATL-Beispiel: Transformation

```
rule SingleValuedDataTypeAttribute2Column {  
from  
  a : Class!Attribute (  
    a.type.ocllsKindOf(Class!DataType) and not a.multiValued  
  )  
to  
  out : Relational!Column (  
    name <- a.name,  
    type <- a.type  
  )  
}
```

484 / 504

For each single-valued Attribute of the type Class, a new Column has to be created.

- Its name has to be set to the attribute's name concatenated with 'id'.
- Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.

Nicht gezeigt: Regeln für multivariate Attribute. Siehe <http://www.eclipse.org/m2m/at1/at1Transformations/Class2Relational/ExampleClass2Relational%5Bv00.01%5D.pdf>

Modell-zu-Text-Transformationen

Arten generierten Texts aus Quellmodellen:

- Programmcode
- Konfigurationsdateien
- Dokumentation wie z.B. Javadoc

Varianten von Modell-zu-Text-Transformationen:

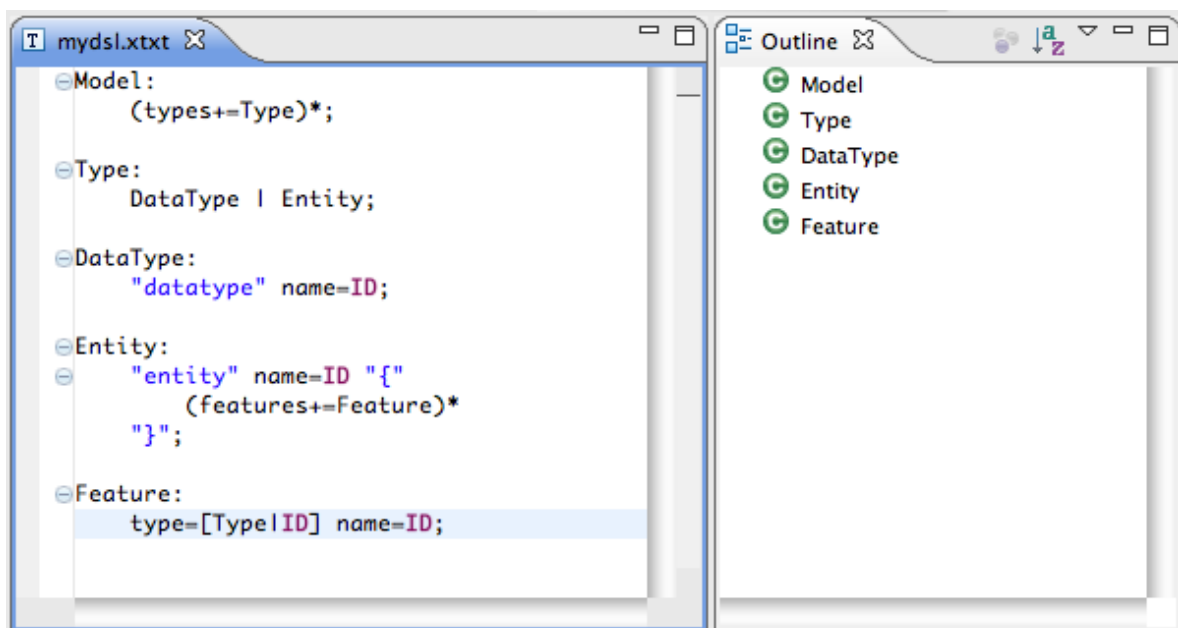
- Visitor-basiert
- Template-basiert

Template-basierte Werkzeuge:

- Xpand aus dem (Eclipse) Xtext Language Modeling Framework
- AndroMDA
- Java Emitter Templates (JET)

485 / 504

Xpand (Xtext-Grammatik)

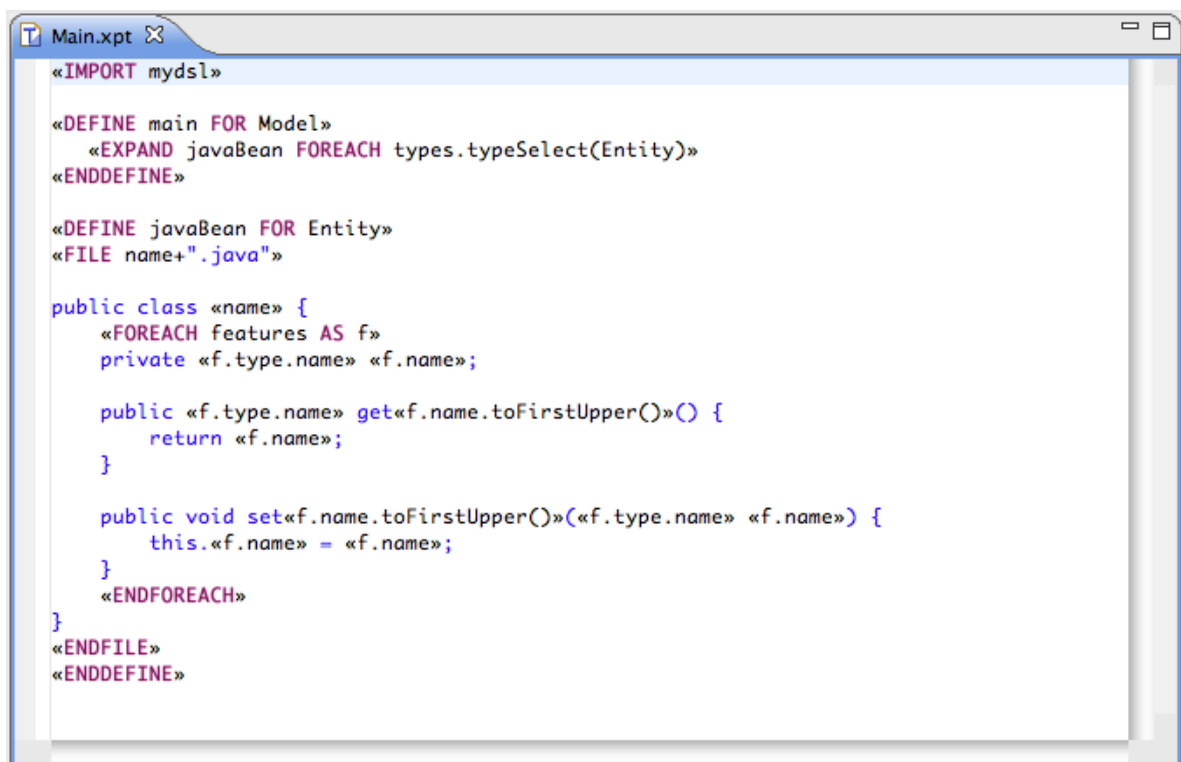


486 / 504

Bildquelle:

http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/xtext_tutorial.html

Xpand (Template)



```
«IMPORT mydsl»

«DEFINE main FOR Model»
  «EXPAND javaBean FOREACH types.typeSelect(Entity)»
«ENDDDEFINE»

«DEFINE javaBean FOR Entity»
«FILE name+ ".java"»

public class «name» {
  «FOREACH features AS f»
  private «f.type.name» «f.name»;

  public «f.type.name» get«f.name.toFirstUpper()»() {
    return «f.name»;
  }

  public void set«f.name.toFirstUpper()»(«f.type.name» «f.name») {
    this.«f.name» = «f.name»;
  }
  «ENDFOREACH»
}
«ENDFILE»
«ENDDDEFINE»
```

Vor- und Nachteile von DSLs

Code-Generierung aus Modellen

- + Arbeitersparnis bei regulären und wohl verstandenen Domänen
- wenigstens eine Referenzimplementierung notwendig
- generierter Code muss mit handgeschriebenem integriert werden
- generierter Code sollte *readonly* sein
- Code-Generatoren müssen erstellt und gewartet werden

Analysefähigkeit

- + abstraktere Darstellung
- „der Generator ist die Spezifikation“
- eigene Analysewerkzeuge notwendig
- eigene Debugging-Werkzeuge notwendig

Wiederholungs- und Vertiefungsfragen

- Was ist die Kernidee der modellgetriebenen Entwicklung? Welche Vorteile verspricht man sich davon?
- Was sind die Merkmale eines Modells im Allgemeinen?
- Was sind Metamodelle und Metametamodelle? Wozu werden sie benötigt?
- Welche Aspekte sind bei der Definition einer Sprache festzulegen?
- Was ist eine domänenspezifische Sprache (DSL)? Was sind die Unterschiede zu einer herkömmlichen Programmiersprache?
- Welche Arten von DSLs unterscheidet man?
- Beschreiben Sie die Unterstützung von EMF für die modellgetriebene Softwareentwicklung.
- Was sind Modelltransformationen und welche Arten gibt es?
- Erläutern Sie konzeptionell Modell-zu-Modell- sowie Model-zu-Text-Transformationen?
- Wie können insbesondere Model-zu-Text-Transformationen realisiert werden?

490 / 504

- 1 Albrecht 1979** ALBRECHT, Alan: Measuring application development productivity. In: Proc. Joint SHARE/GUIDE/IBM Applications Development Symposium, 1979, S. 83–92
- 2 Balzert 1997** BALZERT, Helmut: Lehrbuch der Software-Technik. Spektrum Akademischer Verlag, 1997. – ISBN 3827400651
- 3 Balzert 2008** BALZERT, Helmut: Lehrbuch der Softwaretechnik – Softwaremanagement. 2. Spektrum, Akademischer Verlag, 2008. – ISBN 978-3-8274-1161-7
- 4 Banker u. a. 1991** BANKER, R. ; KAUFFMANN, R. ; KUMAR, R.: An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment. In: Journal of Management Information Systems 8 (1991), Nr. 3, S. 127–150

491 / 504

- 5 **Basili und Weiss 1984** BASILI, R. ; WEISS, D. M.: A Methodology for Collecting Valid Software Engineering Data. In: IEEE Transactions on Software Engineering 10 (1984), November, Nr. 6, S. 728–738
- 6 **Basili und Turner 1975** BASILI, V. ; TURNER, J.: Iterative Enhancement: A Practical Technique for Software Development. In: IEEE Transactions on Software Engineering 1 (1975), Dezember, Nr. 4, S. 390–396
- 7 **Bass u. a. 2003** BASS, Len ; CLEMENTS, Paul ; KAZMAN, Rick: Software Architecture in Practice. 2nd ed. Addison Wesley, 2003
- 8 **Beck 2000** BECK, Kent: Extreme Programming Explained. Addison-Wesley, 2000 (The XP Series). – ISBN 201-61641-6
- 9 **Boehm 1981** BOEHM, B.: Software Engineering Economics. Prentice Hall, 1981
- 10 **Boehm und Turner 2003** BOEHM, B. ; TURNER, R.: Using risk to balance agile and plan-driven methods. In: IEEE Computer 36 (2003), Juni, Nr. 6, S. 57–66

492 / 504

- 11 **Boehm 1979** BOEHM, Barry W.: Guidelines for verifying and validating software requirements and design specification. In: EURO IFIP 79, 1979, S. 711–719
- 12 **Boehm 1988** BOEHM, Barry W.: A spiral model of software development and enhancement. In: IEEE Computer 21 (1988), Mai, Nr. 5, S. 61–72
- 13 **Boehm u. a. 2000** BOEHM, Barry W. ; ABTS, Chris ; BROWN, A. W. ; CHULANI, Sunita ; CLARK, Bradford K. ; HOROWITZ, Ellis ; MADACHY, Ray ; REIFER, Donald ; STEECE, Bert: Software Cost Estimation with COCOMO II. Prentice Hall, 2000
- 14 **Bortz und Döring 2006** BORTZ, Jürgen ; DÖRING, Nicloa: Forschungsmethoden und Evaluation. vierte Auflage. Springer, 2006. – ISBN 978-3-540-33305-0
- 15 **Bortz u. a. 2008** BORTZ, Jürgen ; LIENERT, Gustav A. ; BÖHNKE, Klaus: Verteilungsfreie Methoden in der Biostatistik. zweite Ausgabe. Springer Verlag, 2008. – ISBN 3540675906

493 / 504

- 16 Brooks 1995** BROOKS, Frederick P.:
The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition
2. Addison-Wesley Professional, 1995. – ISBN 0201835959
- 17 Bunse und von Knethen 2002** BUNSE, Christian ; KNETHEN, Antje von: Vorgehensmodelle kompakt. Spektrum-Akademischer Verlag, 2002. – ISBN 978-3827412034
- 18 Buschmann u. a. 1996** BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: Pattern-oriented Software Architecture: A System of Patterns. Bd. 1. Wiley, 1996
- 19 Chidamber und Kemerer 1994** CHIDAMBER, S.R. ; KEMERER, C.F.: A Metrics Suite for Object Oriented Design. In: IEEE Transactions on Software Engineering 20 (1994), Nr. 6, S. 476–493
- 20 Christensen 2007** CHRISTENSEN, Larry B.: Experimental Methodology. 10th edition. Pearson International Edition, 2007. – ISBN 0-205-48473-5
- 494 / 504
- 21 Clements und Northrop 2001** CLEMENTS, Paul ; NORTHROP, Linda M.: Software Product Lines : Practices and Patterns. Addison Wesley, August 2001. – ISBN 0201703327
- 22 Cobb und Mills 1990** COBB, R. H. ; MILLS, H. D.: Engineering Software Under Statistical Quality Control. In: IEEE Software 7 (1990), Nr. 6, S. 44–54
- 23 Dzidek u. a. 2008** DZIDEK, Wojciech J. ; ARISHOLM, Erik ; BRIAND, Lionel C.: A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance. In: IEEE Transactions on Software Engineering 34 (2008), May/June, Nr. 3
- 24 Endres und Rombach 2003** ENDRES, Albert ; ROMBACH, Dieter: A Handbook of Software and Systems Engineering. Addison Wesley, 2003

- 25 **Favre 2004** FAVRE, J. M.: Foundations of Meta-Pyramids: Languages vs. Metamodels - Episode II: Story of Thotus the Baboon. In: BÉZIVIN, J. (Hrsg.) ; HECKEL, R. (Hrsg.): Language Engineering for Model-Driven Software Development Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (Veranst.), 2004
 - 26 **Fenton und Pfleeger 1998** FENTON, N. ; PFLEEGER, S.: Software Metrics: A Rigorous & Practical Approach. 2nd. London : International Thomson Computer Press, 1998
 - 27 **Gamma u. a. 2003** GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: Design Patterns—Elements of Reusable Object-Oriented Software. Addison Wesley, 2003
 - 28 **Garlan u. a. 1995** GARLAN, D. ; ALLEN, R. ; OCKERBLOOM, J.: Architectural Mismatch: Why Reuse is So Hard. In: IEEE Software 12 (1995), November, Nr. 6, S.17–26
 - 29 **Gilb 1988** GILB, Tom: Principles of Software Engineering Management. Harlow, UK : Addison-Wesley, 1988
- 496 / 504
- 30 **Gornik 2001** GORNIK, David: IBM Rational Unified Process: Best Practices for Software Development Teams / IBM Rational Software. 2001 (TP026B, Rev 11/01). – White Paper
 - 31 **Halstead 1977** HALSTEAD, Maurice H.: Elements of Software Science. In: Operating, and Programming Systems Series 7 (1977)
 - 32 **Hofmeister u. a. 2000** HOFMEISTER, Christine ; NORD, Robert ; SONI, Dilip: Applied Software Architecture. Addison Wesley, 2000 (Object Technology Series)
 - 33 **Humphrey 1995** HUMPHREY, Watts S.: A Discipline For Software Engineering. Addison-Wesley, 1995 (SEI series in software engineering)
 - 34 **IBM 1985** : Die Function-Point-Methode: Eine Schätzmethode für IS-Anwendungsprojekte. IBM-Form GE12-1618-1. 1985
 - 35 **Jones 1995** JONES, Capers: Backfiring: Converting Lines of Code to Function Points. In: IEEE Computer 28 (1995), November, Nr. 11, S. 87–88

- 36 Jones 1996** JONES, Capers: Software Estimating Rules of Thumb. In: IEEE Computer 29 (1996), March, Nr. 3, S. 116–118
- 37 Kauffman und Kumar 1993** KAUFFMAN, R. ; KUMAR, R.: Modeling Estimation Expertise in Object Based ICASE Environments / Stern School of Business, New York University. Januar 1993. – Report
- 38 Kemerer 1987** KEMERER, Chris F.: An Empirical Validation of Software Cost Estimation Models. In: Comm. ACM 30 (1987), May, Nr. 5
- 39 Kemerer und Porter 1992** KEMERER, Chris F. ; PORTER, Benjamin S.: Improving the Reliability of Function Point Measurement: An Empirical Study. In: TSE 18 (1992), Nov., Nr. 11
- 40 Kleppe 2008** KLEPPE, A: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley, Pearson Education Inc., 2008

498 / 504

- 41 Kneuper 2006** KNEUPER, Ralf: CMMI – Verbesserung von Softwareprozessen mit Capability Maturity Model. 2. dpunkt.verlag, 2006. – ISBN 3-89864-373-5
- 42 Knight und Leveson 1986** KNIGHT, J.C. ; LEVESON, N.G.: An Experimental Evaluation of the Assumption of Independence in Multiversion Programming. In: IEEE Transactions on Software Engineering 12 (1986), Januar, Nr. 1, S. 96–109
- 43 Kruchten 1998** KRUCHTEN, Phillipe: The Rational Unified Process: An Introduction. Reading, Mass.: Addison-Wesley, 1998
- 44 Lienert 1973** LIENERT, G.A.: Verteilungsfreie Methoden in der Biostatistik. Meisenheim am Glan, Germany : Verlag Anton Hain, 1973. – wird leider nicht mehr aufgelegt
- 45 Ludewig und Lichter 2006** LUDEWIG, Jochen ; LICHTER, Horst: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. dpunkt.verlag, 2006
- 46 McCabe 1976** MCCABE, T.: A Software Complexity Measure. In: IEEE Transactions on Software Engineering 2 (1976), Nr. 4, S. 308–320

499 / 504

- 47 Mernik u. a. 2005** MERNIK, M. ; HEERING, J. ; SLOANE, A. M.: When and how to develop domain-specific languages. In: ACM Computing Surveys 37 (2005), Nr. 4, S. 316–344
- 48 Mills u. a. 1987** MILLS, H.D. ; DYER, M. ; LINGER, R.: Cleanroom Software Engineering. In: IEEE Software 4 (1987), September, Nr. 5, S. 19–25
- 49 Moore u. a. 2009** MOORE, David S. ; MCCABE, George P. ; CRAIG, Bruce A.: Introduction to the Practice of Statistics. sixth edition. W.H. Freeman and Company, 2009
- 50 Müller 2006** MÜLLER, Matthias M.: Do Programmer Pairs make different Mistakes than Solo Programmers? In: Conference on Empirical Assessment In Software Engineering, April 2006
- 51 OSGI Alliance** OSGI ALLIANCE: OSGI. – URL <http://www.osgi.org/Main/HomePage>
- 52 Parker 1995** PARKER, Burton G.: Data Management Maturity Model. July 1995
- 500 / 504
- 53 Pichler 2008** PICHLER, Roman: Scrum — Agiles Projektmanagement erfolgreich einsetzen. dpunkt.verlag, 2008. – ISBN 978-3-89864-478-5
- 54 Poensgen und Bock 2005** POENSGEN, Benjamin ; BOCK, Bertram: Die Function-Point-Analyse. Ein Praxishandbuch. Dpunkt Verlag, 2005. – ISBN 978-3898643320
- 55 Prechelt 2001** PRECHELT, Lutz: Kontrollierte Experimente in der Softwaretechnik – Potenzial und Methodik. Springer, 2001
- 56 Pressman 1997** PRESSMAN, Roger: Software Engineering – A Practioner's Approach. Vierte Ausgabe. McGraw-Hill, 1997
- 57 Reussner und Hasselbring 2009** REUSSNER, R. (Hrsg.) ; HASSELBRING, W (Hrsg.): Handbuch der Software-Architektur. zweite Ausgabe. dpunkt.verlag, 2009
- 58 Royce 1970** ROYCE, W.: Managing the Development of Large Software Systems. In: Proceedings Westcon, IEEEPress, 1970, S. 328–339

- 59 Schmidt u. a. 2000** SCHMIDT, Douglas ; STAL, Michael ; ROHNERT, Hans ; BUSCHMANN, Frank: Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects. Bd. 2. Wiley, 2000
- 60 Siviyy u. a. 2007** SIVIY, Jeannine M. ; PENN, M. L. ; STODDARD, Robert W.: CMMI and Six Sigma – Partners in Process Improvement. Addison-Wesley, 2007 (SEI Series in Software Engineering). – ISBN 978-0-321-51608-4
- 61 Sneed 2004** SNEED, Harry: Vorlesungsskriptum Software-Engineering. Uni Regensburg: Wirtschaftsinformatik (Veranst.), 2004
- 62 Sommerville 2004** SOMMERVILLE, Ian: Software Engineering. Addison-Wesley, 2004
- 63 Stachowiak 1973** STACHOWIAK, H: Allgemeine Modelltheorie. Springer, 1973

502 / 504

- 64 Stahl u. a. 2007** STAHL, Thomas ; VÖLTER, Markus ; EFFTINGE, Sven ; HAASE, Arno: Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management. zweite Auflage. dpunkt.verlag, 2007
- 65 Symons 1988** SYMONS, C. R.: Function Point Analysis: Difficulties and Improvements. In: TSE 14 (1988), Jan., Nr. 1, S. 2–11
- 66 Szyperski u. a. 2002** SZYPERSKI, Clemens ; GRUNTZ, Dominik ; MURER, Stephan: Component Software. Second edition. Addison-Wesley, 2002. – ISBN 0-201-74572-0
- 67 Tichy 1998** TICHY, Walter: Should computer scientists experiment more? In: IEEE Computer 31 (1998), Mai, Nr. 5, S. 32–40
- 68 Winner u. a. 1991** WINNER, B.J. ; BROWN, Donald R. ; MICHELS, Kenneth M.: Statistical Principles in Experimental Design. 3rd edition. McGraw-Hill, 1991 (Series in Psychology)

503 / 504

- 69 Wohlin u. a. 2000** WOHLIN, Claes ; RUNESON, Per ; MAGNUS C. OHLSSON, Martin H. und ; REGNELL, Björn ; WESSLÉN, Anders: Experimentation in Software Engineering – An Introduction. Kluwer Academic Publishers, 2000. – ISBN 0-7923-8682-5
- 70 Yin 2003** YIN, Robert K.: Applied Social Research Methods Series. Bd. 5: Case Study Research. 3rd edition. SAGE Publications, 2003. – ISBN 0-7619-2553-8